THE BURR-BROWN PCI-20000
PERSONAL COMPUTER INTELLIGENT INSTRUMENTATION SYSTEM

PCI-20091W-1

HIGH SPEED ANALOG INPUT BOARD

USER MANUAL

855M180 1.3

Copyright 1989 by Burr-Brown Corporation, Tucson, Arizona, USA
Copyright 1989 by Intelligent Instrumentation, Inc.,
Tucson, Arizona, USA
All rights reserved.

Warranty and Repair Policy Summary

All Burr-Brown products are backed by a warranty policy which is fully described on the back of the packing list enclosed with product shipments. The factory must be contacted prior to the return of any product. If a product is suspected of being inoperative, please contact your local Burr-Brown representative. If the returned unit is under warranty, and requires repair, it will be repaired or replaced, at the discretion of Burr-Brown, and returned to you. If the unit is out of warranty, and requires repair, a repair or replacement price will be quoted. Your purchase order for these repair charges will authorize repair/replacement action. If a unit is returned that is found not to be defective, you may be charged a nominal fee for the evaluation service and for the return shipment.

PCI-20091W-1 USER'S MANUAL
REVISION HISTORY

Version	Date	Revision
1.0	880614	Original
1.1	880909	Added measurement of Fi on pacer clock.
1.2	880930	Added DMA Mode 4 and frequency reference information. Added DMA procedures and made minor corrections.
1.3	891020	ECO 89091902 Major modifications to software section. [MM,SCT]

USE OF EQUIPMENT

Burr-Brown Corp. assumes no responsibility for any direct, indirect or consequential loss or damages resulting from misuse of the equipment or for improper or inadequate maintenance of the equipment or for any such damage or loss resulting from the use of other equipment, attachments, accessories, and repairs at any time made to or placed upon the equipment or any replacement thereof. Furthermore, Burr-Brown Corp. makes no representations or warranties, either expressed or implied, in connection with the use of the equipment in the event it is improperly used, repaired or maintained.

WARNING

Lethal voltages exist inside computers. Always ensure that power is removed before opening the case. Only qualified technicians should install, modify, or adjust equipment inside any computer unit.

CAUTION

Failing to turn off the power when inserting or removing boards will damage the boards and possibly the computer as well.

FCC Radio Frequency Interference Statement

This equipment generates and uses radio frequency energy, and may cause interference to radio or television reception.

Per FCC rules, Part 15, Subpart J, operation of this equipment is subject to the conditions that no harmful interference is caused and that interference must be accepted that may be caused by other incidental or restricted radiation devices, industrial, scientific or medical equipment, or from any authorized radio user.

The operator of a computing device may be required to stop operating his device upon a finding that the device is causing harmful interference and it is in the public interest to stop operation until the interference problem has been corrected.

The user of this equipment is responsible for any interference to radio or television reception caused by the equipment. It is the responsibility of the user to correct such interference.

Table of Contents

Preface

CHAPTER 1: INTRODUCTION

1.1 Manual and Chapter Overview 1.1
1.2 PCI-20091W-1 High-Speed Analog Input Board Description 1.1
1.2.1 Summary 1.1
1.2.2 DEFINITIONS AND CAPABILITIES 1.2
1.2.3 MINIMUM REQUIREMENTS FOR USING THIS PRODUCT 1.2
1.3 PCI System Compatibility and Emulation 1.3

CHAPTER 2: HARDWARE CONFIGURATION AND INSTALLATION

2.1 Configuration and Installation—An Overview 2.1
2.2 Setting the Board Addresses 2.1
2.3 Analog Input Voltage Ranges 2.6
2.4 Start of Conversion 2.6
2.5 Channel Scanning Sequence 2.7
2.6 Interrupt Levels and Sources 2.7
2.7 DMA Control 2.9
2.7.1 DMA CAPABILITY AND INTERRUPT STATUS 2.9
2.7.2 DMA MODES 2.9
2.7.3 DMA PACING SOURCE 2.10
2.7.4 DELAY COUNTER 2.10
2.7.5 DMA CHANNEL SELECTION 2.10
2.8 Rate Generator 2.10
2.9 Installing the Board Into Your Computer 2.11
2.10 Pins and Ports: Connecting to the Outside World 2.12
2.10.1 ANALOG CONNECTOR (P2) 2.12
2.10.2 DIGITAL CONNECTOR (P3) 2.12

CHAPTER 3: SOFTWARE INSTALLATION

3.1 Chapter Overview 3.1
3.2 Software Drivers Included With Your System 3.1
3.3 System Configuration Requirements 3.2
3.4 The PCI-20092S Software Drivers License Agreement 3.2
3.5 Creating a Working Copy of Your Software 3.2
3.5.1 PROCEDURE FOR ONE OR TWO FLOPPY DISK DRIVES 3.3
3.5.2 PROCEDURE FOR ONE FLOPPY DISK DRIVE AND A FIXED DISK DRIVE . . . 3.4
3.6 Software Drivers Description 3.5
3.7 Loading the Software Drivers Routines 3.8
3.7.1 ERRORS WHILE LOADING THE SOFTWARE DRIVERS 3.9
3.7.2 MEMORY RESIDENT PROGRAMS 3.10
3.7.3 SWITCHES 3.11
SWITCH /V (INTERRUPT VECTOR) 3.11
SWITCH /G (BYPASSING CONFIRMATION) 3.11
SWITCH /B (ALLOCATE DMA BUFFER) 3.11
SWITCH /X (EXECUTE COMMAND LINE) 3.12
3.7.4 EXAMPLES OF LOADING THE SOFTWARE DRIVERS 3.12

CHAPTER 4: HIGH SPEED ANALOG DATA ACQUISITION

4.1 Introduction and Chapter Overview 4.1
4.2 Hardware Configuration 4.1
4.3 Programming Guidelines 4.2
4.3.1 TIMING 4.2
4.3.2 TESTING HIGH SPEED SAMPLING RATES 4.3
4.4 High Speed Mode 4: Hardware-Controlled High Speed Analog Acquisition 4.3
4.4.1 DESCRIPTION 4.3
4.4.2 PROGRAMMING 4.4

CHAPTER 5: DMA DATA ACQUISITION AND CONTROL

5.1 Introduction and Chapter Overview 5.1
5.1.1 DMA INTRODUCTION 5.1
5.1.2 SEQUENCE OF OPERATION 5.1
5.1.3 INFORMATION REQUIRED FOR RUNNING DMA 5.2
5.2 DMA Mode 2. Start on Command; Stop on Terminal Count 5.2
5.3 DMA Mode 3. Start on Command; Stop on Trigger after Delay 5.2
5.4 DMA Mode 4. Start on Command; Stop on Command. 5.3
5.5 DMA Rates 5.3
5.6 DMA and Foreground Programs 5.4
5.7 DMA Buffers 5.4
5.7.1 BUFFER SIZE 5.5
5.7.2 BUFFER DATA ORGANIZATION 5.5
5.8 Data Frames and Clusters 5.5
5.8.1 AUTO-SEQUENCED CHANNELS 5.6
5.8.2 DATA FRAMES AND DMA MODES 3 and 4 . . . . 5.6
5.9 I/O Types 5.6
5.10 DMA INSTRUCTIONS 5.7
5.10.1 DMA INITIALIZATION INSTRUCTIONS 5.7
5.10.2 CONTROL INITIALIZATION INSTRUCTIONS 5.7
5.10.3 DMA EXECUTION INSTRUCTIONS 5.7
5.10.4 BUFFER CONTROL INSTRUCTIONS 5.7

CHAPTER 6: SOFTWARE TECHNICAL REFERENCE

6.1 Chapter Overview 6.1
6.2 Creating Application Programs 6.1
6.3 Hardware Initialization 6.3
6.3.1 INITIALIZATION SEQUENCE 6.3
6.3.2 EXAMPLES OF HARDWARE INITIALIZATION 6.3
Example 1: One PCI-20091W-1 . . . . 6.3
Example 2: Two PCI-20091W-1 Boards 6.4
Example 3: PCI-20091W-1 with a PCI-20041C-3A Carrier 6.4
6.4 Software Calls 6.4
6.4.1 UTILITY INSTRUCTIONS 6.4
6.4.2 CONFIGURATION INSTRUCTIONS 6.5
6.4.3 READ INSTRUCTIONS 6.5
6.4.4 WRITE INSTRUCTIONS 6.5
6.4.5 BUFFER UTILITIES 6.5
6.4.6 HIGH SPEED ANALOG ACQUISITION INSTRUCTIONS 6.5
6.4.7 DMA DATA ACQUISITION AND CONTROL INSTRUCTIONS 6.6
6.5 Instruction Specifications 6.6

BUF.DEC (Decode Buffer) . . . . . Buffer Instruction 6.8
CNF.DMA (Configure DMA) . . . . . Configuration Instruction 6.9
CNF.HS (Configure High Speed) . . Configuration Instruction 6.11
CNF.RG (Configure Rate Generator) Configuration Instruction 6.13
DMA.RUN (Run DMA) . . . . . DMA Control Instruction 6.15
DMA.STAT (DMA Status) . . . . . DMA Control Instruction 6.16
DMA.STOP (Stop DMA) . . . . . DMA Control Instruction 6.18
ERR.SYS (System Error) . . . . . Utility Instruction 6.19
HS.RUN (Run High Speed) . . . . . Data Acquisition Instruction 6.20
INIT (Initialize) . . . . . Utility Instruction 6.21
READ.CH (Read Channel) . . . . . Read Instruction 6.22
SYSINIT (System Initialize) . . . . Utility Instruction 6.23
WRITE.CH (Write Channel) . . . . . Write Instruction 6.24
6.6 Software Error Codes 6.25
6.6.1 0 - 99. System Error Codes 6.25
6.6.2 100 - 199. Analog Input Error Codes 6.26
6.6.3 800 - 899. Rate Generator Error Codes 6.26
6.6.4 15,000 - 15,799. High Speed Acquisition Error Codes 6.26
6.6.5 15,800 - 16,499. DMA Error Codes 6.28
6.6.6 20,100 - 20,199. Buffer Error Codes 6.29

CHAPTER 7: HARDWARE TECHNICAL REFERENCE

7.1 Introduction and Chapter Overview 7.1
7.2 Testing and Calibration 7.1
7.3 Doing It Yourself: Hardware Programming 7.3
7.3.1 ANALOG-TO-DIGITAL (A/D) CONVERTER 7.3
7.3.2 RATE GENERATOR 7.3
7.3.3 DMA OPERATION 7.4
7.3.4 BOARD IDENTIFICATION 7.4
7.4 Register Offsets 7.4
7.4.1 I/O REGISTERS 7.4
7.4.2 MEMORY ADDRESSES 7.4
7.4.3 REGISTER OFFSETS 7.5
7.4.4 REGISTER USE 7.6
Offset 000: Board ID; Interrupt/Status Mode 7.6
Offset 001: System Status; Interrupt/Status Clear 7.7
Offset 002: DMA Start; DMA Mode 7.7
Offset 003: DMA Stop 7.8
Offset 004: ID Code 2 . . . . . 7.8
Offsets 005 - 03F: Reserved 7.8
Offset 040: Low Order Rate Generator Counter 7.8
Offset 041: High Order Rate Generator Counter 7.8
Offset 042: Delay Counter 7.9
Offset 043: Rate Generator and Delay Counter Control 7.9
Offsets 044 - 0FF: Reserved 7.11
Offset 100: ID Code 3 . . . . . 7.11
Offset 103: Start Conversion 7.12
Offset 104: Input Channel 7.13
Offset 105: Scanner Start Channel 7.13
7.5 Hardware Programming Procedures 7.13
7.5.1 MEMORY ADDRESSES 7.13
7.5.2 ANALOG INPUT PROCEDURE 7.13

7.5.3 DATA ACQUISITION WITH HARDWARE TIMING 7.14
7.5.4 DELAY COUNTER INITIALIZATION PROCEDURE 7.14
7.5.5 DELAY COUNTER ENABLE OR DISABLE PROCEDURE 7.15
7.5.6 DELAY COUNTER READ PROCEDURE 7.15
7.5.7 RATE GENERATOR INITIALIZATION PROCEDURE 7.16
7.5.8 RATE GENERATOR ENABLE OR DISABLE PROCEDURE 7.16
7.5.9 DMA PROCEDURES 7.16
7.5.10 DMA MODE 2 PROCEDURE 7.16
7.5.11 DMA MODE 3 PROCEDURE 7.17
7.5.12 DMA MODE 4 PROCEDURE 7.18
7.6 Jumper Definitions Summary 7.19
7.7 Hardware Specifications 7.20

Figure Listing

Fig. 1.1 Hardware Block Diagram 1.4
Fig. 2.1 IBM Personal Computer Memory Map 2.3
Fig. 2.2 Examples of setting addresses 2.4
Fig. 2.3 Jumper Locations 2.5
Fig. 2.4 Connector pin assignments 2.13
Fig. 3.1 BASIC Language Software Drivers System 3.6
Fig. 3.2 PCIHEADQ.BAS listing 3.7

Table Listing

TABLE 2.1Analog Input Voltage Range Settings 2.6
TABLE 2.2 Start of Conversion Jumper Settings 2.7
TABLE 2.3 Channel Scanning Sequence Jumper Settings 2.7
TABLE 2.4 Interrupt Source Jumper Settings 2.8
TABLE 2.5 Interrupt Level Jumper Settings 2.8
TABLE 2.6 DMA ControlJumper Settings 2.9
TABLE 2.7 DMA Mode Descriptions 2.10
TABLE 2.8 DMA Control Jumper Settings 2.10
TABLE 2.9 Digital Connector P3 pinouts and signal descriptions 2.12
TABLE 3.1 Summary of PCI_92S.EXE switches 3.12
TABLE 4.1 High Speed Programming Instructions 4.2
TABLE 5.1 Effect of the DMA Process on Foreground Programs 5.4
TABLE 6.1 I/O Types 6.7
TABLE 6.2 DMA Status Return Values 6.16
TABLE 7.1 PCI-20019M-1A Calibration Set Points 7.2
TABLE 7.2 PCI-20019M-1A Calibration Set Points 7.2
TABLE 7.3 Register Offsets 7.5
TABLE 7.4 DMA Control Jumper Settings 7.6
TABLE 7.5 Offset 043 Control Words 7.10
TABLE 7.6 Offset 043 Rate Generator (RG) Programming Codes 7.10
TABLE 7.7 Offset 043 Counter Data and Status Read Command Format 7.11
TABLE 7.8 Offset 040 or 041 Returned Status Format 7.11
TABLE 7.9: Jumper Definitions Summary 7.19
TABLE 7.10 A/D Conversion Formaulas 7.22
TABLE 7.11 A/D Codes

Preface

The PCI-20000 Personal Computer Instrumentation System

The PCI-20000 System is a modular instrumentation system for use with IBM PC(tm) and compatible personal computers. The System consists of **Instrument Modules** for data acquisition and control, **Carriers** to provide an interface between the Modules and the PC bus, **Termination Panels** and accessories to facilitate bringing "real world" signals to and from the Modules, and **Software Support** packages to enable the user to operate his system easily and effectively.

The **Instrument Modules** are small 4" X 4" cards, each one having a specific set of capabilities. Modules may be chosen to provide such functions as digital inputs/outputs, analog inputs, analog outputs, triggering, and frequency measurements. Up to three Modules may be plugged into a **Carrier**, which is a 13" X 4" card designed to interface with the IBM PC(tm) bus. All Carriers provide inter-Module communications and power for the Instrument Modules, and some Carriers also provide on-board digital I/O and rate-generator functions.

Various rack-mountable Analog and Digital **Termination Panels** provide convenient screw-terminal connections for field wiring, including thermocouples and RTDs. Some Panels have board space for user-installed signal-conditioning components; other Panels provide signal-conditioning or isolation features.

Software Support packages provide high-level and assembly language support for the PCI-20000 System. The most useful hardware-level functions of the individual Modules and Carriers are supported, as well as various system configurations, such as multiple-channel high-speed analog acquisition.

Each of the components of the PCI-20000 System is fully described in its own User's Manual. In addition, a PCI-20000 System User's Manual is provided to explain how to put a system together and install it in a computer. The System Manual also contains useful tutorial information on computer and data-acquisition concepts. We recommend that you become familiar with the PCI-20000 System by reading the System Manual first, and then the User's Manual for each of the components you have chosen for your application.

CHAPTER 1: INTRODUCTION

1.1 Manual and Chapter Overview

Congratulations on purchasing another quality member of the PCI family of products. You will find that this product provides you with an easy-to-use high-speed analog input system capable of supporting high speed and direct memory access (DMA) data acquisition techniques.

This manual is organized into seven chapters. The first chapter is an introduction to the manual, an introduction to the PCI family of products, and an introduction to the product you have just purchased.

Chapter 2 tells you how to install your hardware, and chapter 3 tells you how to install your software.

Chapters 4 and 5 cover the details of high speed data acquisition and Direct Memory Access (DMA) data acquisition.

Chapter 6 is an in-depth, technical discussion of the software that came with your system, along with some programming guidelines for writing your own software for this system.

Chapter 7 is an in-depth, technical discussion of your hardware, including a specifications list.

Appendix A contains the sample program listings.

The last part of this manual is that indispensable tool, the index.

1.2 PCI-20091W-1 High-Speed Analog Input Board Description

1.2.1 Summary

The PCI-20091W-1 High Speed Analog Input Board is a single Board system that provides high speed and DMA analog acquisition functions with up to 8 eight single-ended analog input channels; a 12-bit resolution analog-to-digital (A/D) converter; a general-purpose rate generator that can be used as a data acquisition pacer; and PC interrupt capability. Analog acquisition speeds of up to 89,000 samples per second can be obtained with this Board.

The Board's analog input section provides eight single-ended input channels. By proper placement of jumpers (see chapter 2), channel ranges may be selected for ±2.5 volts, ±5 volts, ± 10 volts, 0 to 5 volts, and 0 to 10 volts.

The analog input system can provide high-speed data acquisition using a variety of conversion pacing sources. These sources include two hardware sources: an External Start Convert, and the onboard Rate Generator; and two software sources: a Software Start Convert command, and Read MSB Data.

You can select the channel sampling sequence through hardware or software. When you choose hardware-controlled channel scanning, the channel is automatically advanced by the hardware following each conversion.

The PCI-20091W-1 provides a rate generator function. The rate generator signal is buffered and brought out to a rate generator output, in addition to being available on the Board as a data acquisition pacing signal. The rate generator output provides a variable duty cycle with a minimum duty cycle of 50%.

The Board accepts external TTL control signals, in addition to the External Start Convert signal mentioned above. A Convert Inhibit input can be used to gate enable High Speed or DMA analog data acquisitions. An External Interrupt input can provide interrupts to the PC from an external source or act as a DMA Trigger event signal. A PC interrupt can be also generated to signal termination of a DMA data acquisition.

See figure 1.1 for a hardware block diagram of the PCI-20091W-1.

1.2.2 DEFINITIONS AND CAPABILITIES

Programming analog channels, reading analog data, programming the Rate Generator, and setting up the DMA ciruitry can be accomplished through the use of the included PCI-20092S Software Drivers, or the PCI-20026S and PCI-20027S Software Drivers, or by the use of your own custom software. Programming guidelines are provided later in this manual to assist users who wish to write their own software.

The PCI-20092S Software Drivers, described in detail in chapters 3 through 6, are a subset of the PCI-20026S and PCI-20027S Software Drivers. This subset supports the specific functions of the PCI-20091W-1 Board. The PCI-20026S Software Drivers support the functions of other PCI products. They also support the programming languages IBM Interpreted BASIC, C, and Turbo Pascal.

1.2.3 MINIMUM REQUIREMENTS FOR USING THIS PRODUCT

In order to use this product successfully, you need to have at a minimum the following available:

1. The PCI-20091W-1 High Speed Analog Input Board and accessories. We recommend the PCI-20012A-1 or the PCI-20012A-2 Analog Signal Cables, and the PCI-20010T-1 Analog Termination Panel.

2. The accompanying PCI-20092S Software Drivers (packaged with this product).

3. Microsoft QuickBASIC programming language.

4. An IBM PC, XT, AT, or compatible.

This list gives a minimum configuration. This product is flexible and expandable; other products you can use with this system are described in the next section.

1.3 PCI System Compatibility and Emulation

This section describes the PCI-20091W-1's compatibility with the PCI system.

The PCI-20091W-1 Board emulates most functions of a PCI-20041C-3A Carrier (without digital I/O) with a PCI-20019M-1A High Speed Analog Input Module installed in Module Position 1. It does not support all hardware interrupt sources. Using the PCI-20092S Software Drivers, the Board can be combined with one or more other PCI-20091W-1 Boards, or with one or more PCI-20041C Carriers with or without PCI-20019M-1A Modules. (The PCI-20092S Software Drivers will not support digital I/O functions of the PCI-20041C Carriers).

The included PCI-20092S-1 Software Drivers support Microsoft QuickBASIC and the high speed and DMA functions of your Board (see chapters 4 and 5). They also support all functions of the PCI-20091W-1 Board as performed by the PCI-20041C-3A Carrier (without digital I/O) in combination with the PCI-20019M-1A High-Speed Analog Input Module. Please note that the PCI-20041C-3A is more complex than the PCI-20091W-1; it therefore has greater flexibility and more capabilities.

The PCI-20091W-1 Board is compatible with the PCI-20026S Software Drivers, the PCI-20046S Software Drivers (version 3.0 and later), and with all members of the PCI family of hardware products. If you choose to combine the PCI-20091W-1 Board with any PCI hardware products (except the PCI-20041C Carrier or the PCI-20019M Module), you *must* have the PCI-20026S Software Drivers in order to support the functions of each component in the system.

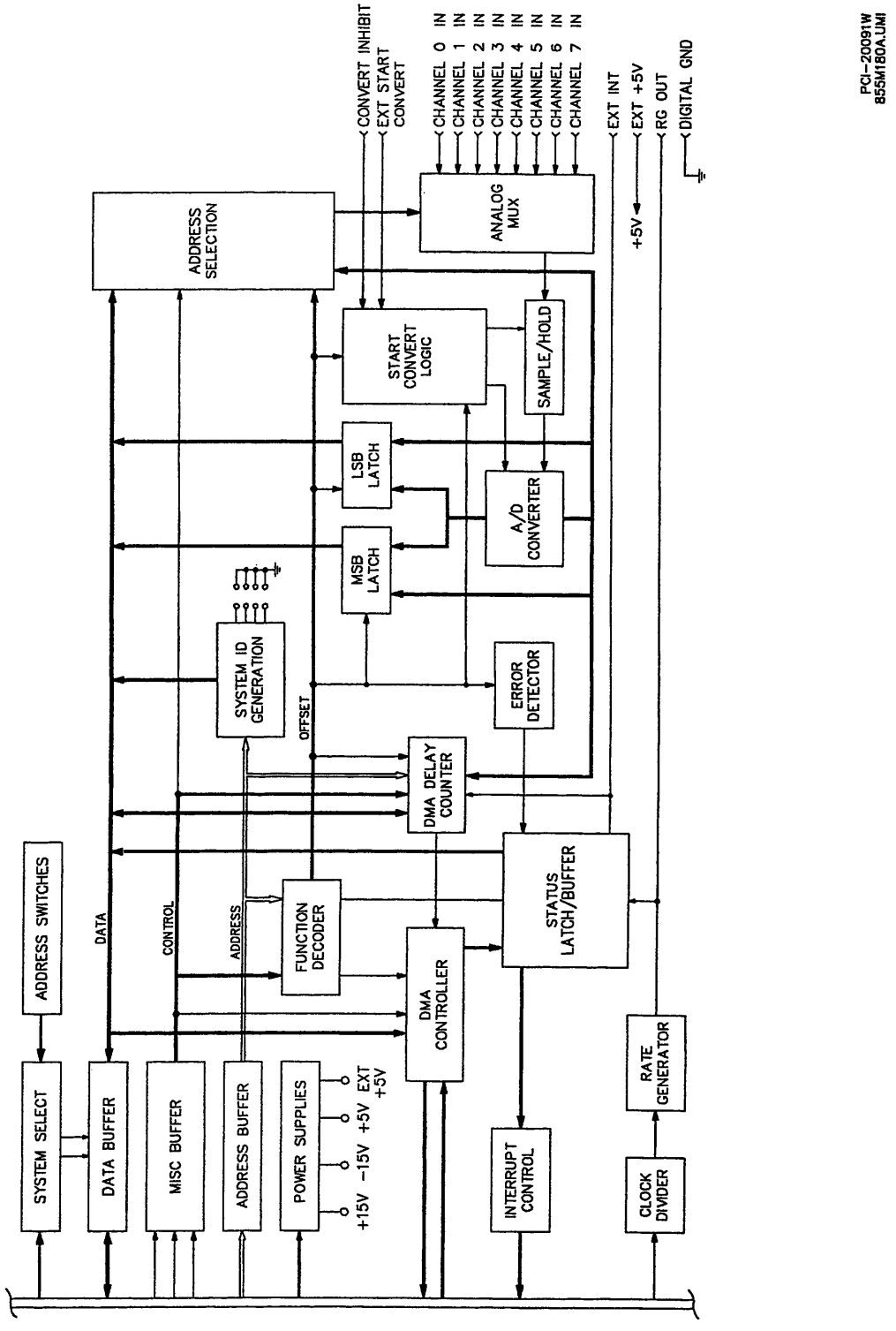


Fig. 1.1 Hardware Block Diagram

CHAPTER 2: HARDWARE CONFIGURATION AND INSTALLATION

2.1 Configuration and Installation—An Overview

This chapter covers physically configuring and installing your hardware. For more in-depth information on the hardware, see chapter 7.

You must set switches and jumpers to configure your Board. The following list gives the configuration options for your board, and the sections in which these topics are covered.

- You must set the base address of your Board (section 2.2).
- You must choose an analog signal range (section 2.3).
- You must select a method of starting a conversion (section 2.4).
- You can enable an automatic channel scanning sequence (section 2.5)
- You can configure your Board to generate interrupts, specifying the interrupt level and sources (section 2.6).
- You can configure your Board for DMA control capability (section 2.7).
- You may make use of the Rate Generator output signal (section 2.8).

For installation, you will need to know how to physically place the Board into your computer, and how to connect your Board to the "outside world." These topics are covered in sections 2.9 and 2.10, respectively.

The locations of the jumpers are shown in figure 2.3. Table xx (chapter 8) summarizes the jumper functions. Only plug-in jumpers are intended to be removed.

2.2 Setting the Board Addresses

In order to use your Board, you must set its base address. If this address is not set properly, the Board will not work at all, and it may prevent other components of your computer system from operating. This section describes how to select an address.

The PCI-20091W-1 system occupies 1 Kbyte of memory space. The base address is selected by a 10-position dual-in-line package (DIP) switch on the board. The range of addresses is from 400 hex (segment 40 hex) to FFC00 hex (segment FFC0 hex) in hex 400 increments.

When you choose the base address for your Board, be sure to select a location in memory that is not used by any other hardware in your system. The particular address you use is determined by the hardware configuration of your system. If your computer includes expanded memory, conforming to the Lotus/Intel/Microsoft

(LIM) specification, we recommend that you configure this memory to start at address D0000 (Hex), in order to ensure the availability of addresses between CD000 and CFC00 for your Board. Refer to figure 2.1 for an illustration of the IBM PC/AT memory map, and the recommended locations for addressing your Board. Factory configuration of the Board's address is CDC0 (segment CDC0 hex).

The base address of the 1-Kbyte block to be used is selected by switches 1 through 10 on the Board, with switch 1 corresponding to address bit 10, and switch 10 corresponding to address bit 19. A switch setting of "OFF" determines a bit value of 1, and a setting of "ON" determines a bit value of 0.

Figure 2.2 illustrates several Board addressing examples, showing how to set the switches for addresses CDC00, CD000, CD400, and CD800 (Hex). Please note that the Board's base address, as set by the switches, is expressed as a five-digit Hex number (20-bit binary number), in which bits 10 through 19 may be switched. The Segment:Offset address that results from the physical switch setting is also shown for each example.

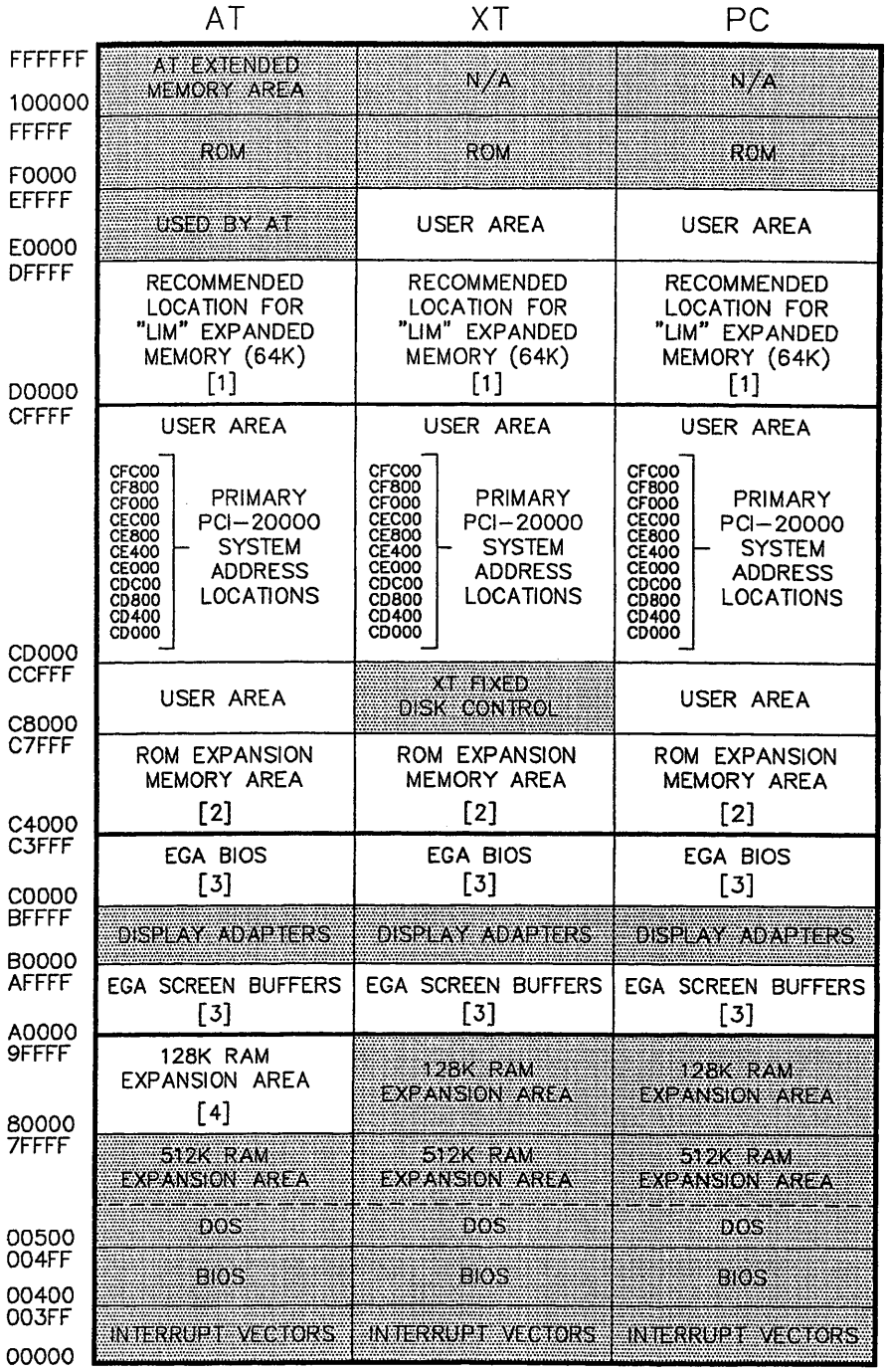
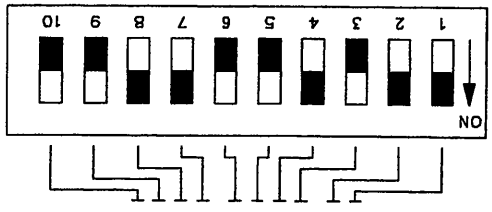


Fig. 2.1 IBM Personal Computer Memory Map

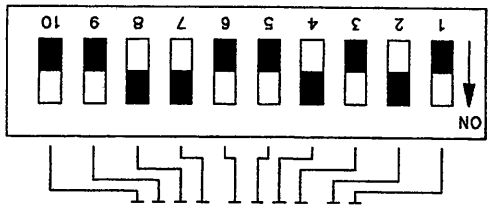
EXAMPLE 1: Memory Address CD00:0000



Address CD000 (Hex) = **1100 1101 00**00 0000 0000 (Binary)

A19 A0

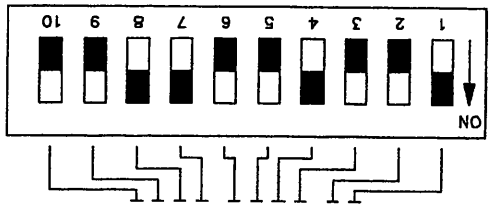
EXAMPLE 2 Memory Address CD40:0000



Address CD400 (Hex) = **1100 1101 01**00 0000 0000 (Binary)

A19 A0

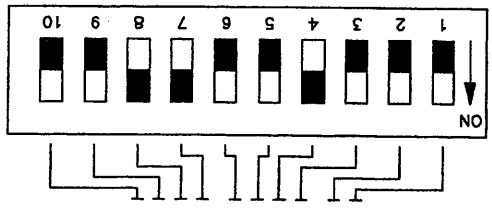
EXAMPLE 3 Memory Address CD80:0000



Address CD800 (Hex) = **1100 1101 10**00 0000 0000 (Binary)

A19 A0

EXAMPLE 4 Memory Address CDC0:0000



Address CDC00 (Hex) = **1100 1101 11**00 0000 0000 (Binary)

A19 A0

PCI-20091W
855M180E.UMI

Fig. 2.2 Examples of setting addresses

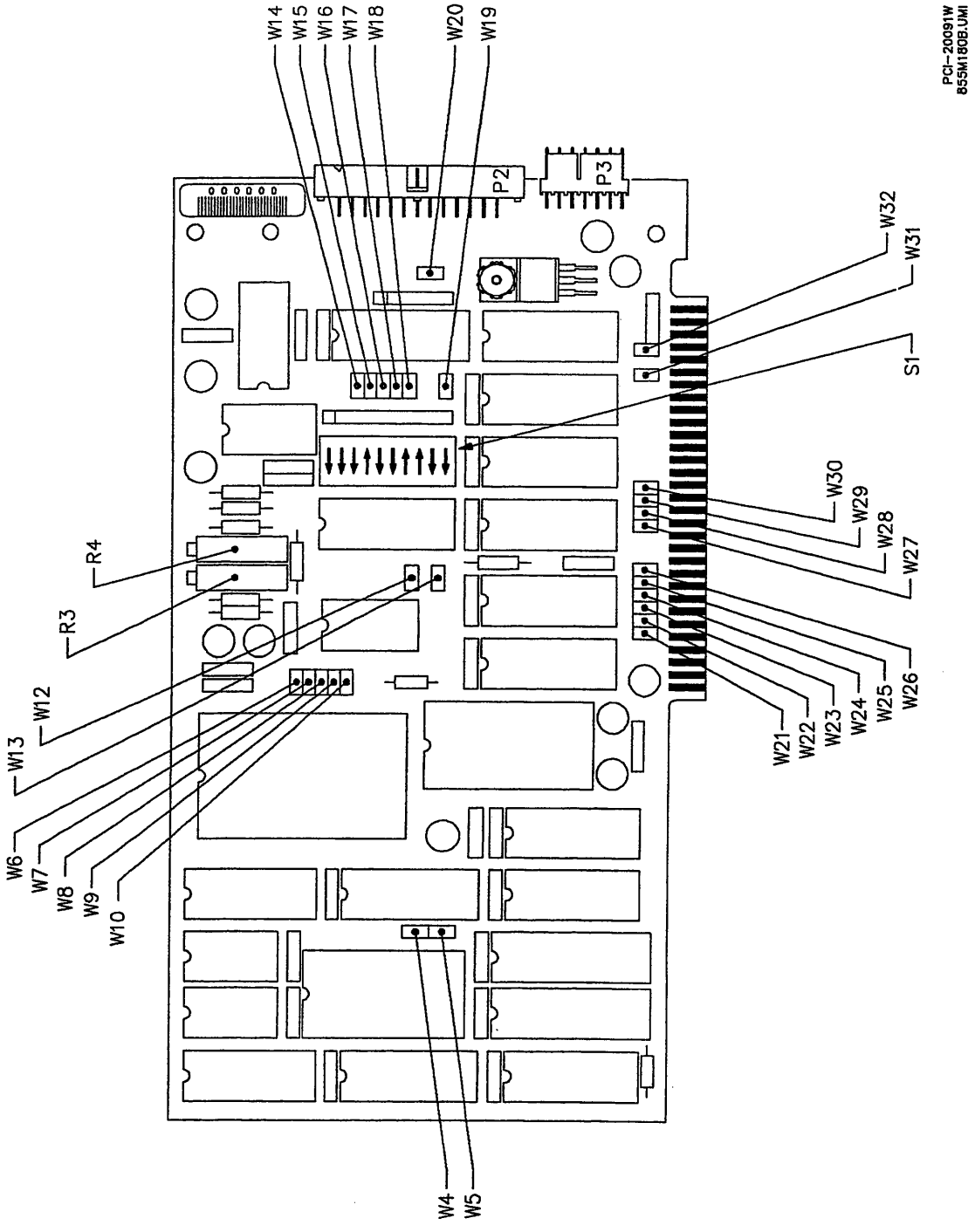


Fig. 2.3 Jumper Locations

2.3 Analog Input Voltage Ranges

An Analog to Digital (A/D) converter on your Board converts the analog signal on the selected channel into digital data to be transferred to the computer. There are five jumper-selectable A/D range/polarity settings. Your Board has been set at the factory for the analog input voltage range of ±10 volts. However, you can easily change this setting by changing the jumpers, as shown in table 2.1. We recommend that you use the smallest range that you think will fit your signal. The digital code for the output data is complementary binary for the unipolar ranges (0 to 5 volt and 0 to 10 volt), and complementary offset binary for the bipolar ranges (±2.5, ±5 and ±10 volts).

TABLE 2.1 Analog Input Voltage Range Jumper Settings

Jumper Settings:

Voltage	Jumpers IN	Jumpers OUT
±2.5 V	W7, W9, W10	W6, W8
±5V	W7, W9	W6, W8, W10
±10V[1]	W6, W9	W7, W8, W10
0 to 5V	W7, W8, W10	W6, W9
0 to 10V	W7, W8	W6, W9, W10

[1] Factory configuration

2.4 Start of Conversion

The PCI-20091W-2 provides four means of starting an A/D conversion: an external signal, the onboard Rate Generator, start conversionwhen the previous data is read, and software command. Conversions can always be started by software command. The jumper settings to select the other three sources are shown in table 2.2.

The *hardware* start of conversion sources are External Start Convert and the onboard Rate Generator. An external start convert signal can be generated by a TTL low-to-high transition at pin 2 of connector P3. The Rate Generator is an counter driven by an on-board 8MHz frequency reference. If the onboard Rate Generator is selected, it must be properly programmed. See section 7.4 for information on how to do this, or use the included PCI-20092S Software Drivers. A hardware start of conversion source must be used for high speed or DMA data acquisition. In either hardware mode, analog conversions may be inhibited by taking the Conversion Inhibit signal (connector P3 pin 3) low. The Board is factory configured for start of conversion from the Rate Generator.

Conversion can also be started when the previous data is read, or by software command. These are *software* sources. A software start convert command is generated by a read or write to a command register, or by a read of the most-significant data byte (MSB). Refer to section 7.4 for a complete discussion of I/O registers. Software start convert is appropriate for acquiring randomly timed channel samples. (Start convert on read of MSB data is included for backward compatibility with other PCI products and is not a recommended method.)

TABLE 2.2 Start of Conversion Jumper Settings

Jumper Settings:

Source	Jumper IN	Jumpers OUT
External Start Convert	W20	W12, W13
Rate Generator[1]	W12	W13, W20
Read of MSB data	W13	W12, W20

[1]Factory configuration

2.5 Channel Scanning Sequence

The input channel may be selected by a hardware channel scanner, or under software control. The hardware scan selects channels **N** through 7 in sequence, where **N** is an integer from 0 to 7. The scanner must be enabled by setting jumpers as shown in table 2.3. The channel is advanced by a signal derived from the analog-to-digital (A/D) converter's end of convert (EOC) line. The input channel may be selected in software by writing the desired channel to a channel-selection register. Programming the channel scanner and channel selection are discussed in section 7.4.

TABLE 2.3 Channel Scanning Sequence Jumper Settings

Jumper Settings:

Channel Sequence Mode	Jumpers IN	Jumpers OUT
Software Selection	W15, W19	W14
Hardware Scan[1]	W14	W15, W19

[1]Factory configuration

2.6 Interrupt Levels and Sources

The PCI-20091W-1 provides a PC interrupt, IRQ0*, the level and source of which are jumper selectable. An interrupt is used to signal the computer that an event has happened, and that the Board requires action from the computer. There is an

excellent discussion of interrupts, and their applications to data acquisition with the PCI-20000 System, in *The Handbook of Personal Computer Instrumentation,* which is provided with your system.

In order to use interrupts from your Board, a signal from the Board must be connected to the Board's IRQ0* line, and the Board's IRQ0* line must be connected to an interrupt level input on the computer. Please refer to tables 2.4 and 2.5 for the jumper settings needed to select interrupt level and source.

The interrupt **source** is selected by jumpers W4 and W5. W4 IN selects the DMA Terminal Count signal as interrupt source. This signal is generated when the Board receives a terminal count from the PC's DMA controller. W5 IN selects the External Interrupt source. Factory configuration for W4 and W5 is OUT. An interrupt on the selected level will occur when any selected source goes low.

TABLE 2.4 Interrupt Source Jumper Settings

Interrupt Source	Jumpers IN	Jumpers OUT
None1	----------	W4, W5
DMA Terminal Count	W4	W5
External Interrupt	W5	W4

[1] Factory configuration

The hardware interrupt **level** on the PCI-20091W-1 is controlled by jumpers W22 through W26 and jumper W32, using the settings given in table 2.5. No more than one of these jumpers should be installed. Factory configuration for these jumpers is OUT.

TABLE 2.5 Interrupt Level Jumper Settings

Interrupt Level	Jumpers IN	Jumpers OTU
None1	---	W22, W23, W24, W25, W26, W32
2	W32	W22, W23, W24, W25, W26
3	W22	W23, W24, W25, W26, W32
4	W23	W22, W24, W25, W26, W32
5	W24	W22, W23, W25, W26, W32
6	W25	W22, W23, W24, W26, W32
7	W26	W22, W23, W24, W25, W32

[1] Factory configuration

2.7 DMA Control

The PCI-20091W-1 incorporates a DMA controller similar to the one used on the PCI-20041C-3A. All DMA transfers are made to the PC computer's memory, and are paced by the A/D converter's end of conversion (EOC) signal. The maximum DMA rate is determined by the PCI-20091W-1 Board's hardware capability as 89,000 samples/sec. Programming the Board for DMA data acquisition can be accomplished through use of the included PCI-20092S Software Drivers. Information on programming the Board directly is contained in section 7.4.

2.7.1 DMA CAPABILITY AND INTERRUPT STATUS

The DMA control and DMA terminal-count interrupt capability of the PCI-20091W-1 Board are enabled by setting jumpers appropriately. These jumpers also determine the ID code that will be read from the Board ID register, Offset 000. (Register offsets are described in chapter 7.) Two jumper settings correspond to "extended mode" configurations. These ID codes are not supported by earlier versions of the software drivers (PCI-20046S/PCI-20047S, and early versions of the PCI-20092S Software Drivers). Terminal-count interrupt modes are not supported by the included PCI-20092S Software Drivers. The DMA control and ID code jumper settings are shown in table 2.6 below.

TABLE 2.6 DMA Control Jumper Settings

ID Code	Jumper IN	Jumpers OUT
DMA without terminal count	---	W16, W17, W18
Non-DMA mode	W16	W17, W18
DMA with terminal count	W17	W16, W18
Extended without terminal count	W16, W17	W18
Extended with terminal count[1]	W18	W16, W17

[1] Factory configuration

2.7.2 DMA MODES

Direct memory access (DMA) is a method of loading output data from the PCI-20091W-1 to the host computer's memory under automatic hardware control. The PCI-20027S Software Drivers support nine different DMA modes. The PCI-20092S Software Drivers, as a subset of the PCI-20047S Software Drivers, support three of those modes: DMA Mode 2, DMA Mode 3, and DMA Mode 4. The different DMA modes are characterized by different start and stop conditions for data acquisition, shown in table 2.7. There is a detailed discussion of DMA, and of these DMA modes, in chapter 5.

TABLE 2.7 DMA Mode Description

DMA Mode	Start Condition	Stop Condition
2	Command	Terminal Count
3	Command	Trigger with Delay
4	Command	Command

2.7.3 DMA PACING SOURCE

The PCI-20091W-1's DMA controller is paced by the A/D converter's End Of Convert signal. The data acquisition pacing is controlled by the source of the A/D converter's start convert signal. This signal can be configured to come from either the Rate Generator or the External Start Convert (see section 2.4).

2.7.4 DELAY COUNTER

DMA Mode 3 permits sampling of pre- and post-trigger data. The delay counter, which can be programmed for a count of up to 65,000 samples, controls the amount of data taken after the trigger event. The Delay Counter must be initialized before DMA Mode 3 is started (see chapter 6, or section 7.4).

2.7.5 DMA CHANNEL SELECTION

The DMA channel is jumper selectable among channels 1, 2, and 3 by jumper W21 and jumpers W27 through W31, as shown in table 2.8.

TABLE 2.8 DMA Control Jumper Settings

DMA Channel	Jumpers IN	Jumpers OUT
None[1]	---	W21, W27, W28, W29, W30, W31
1	W27, W28	W21, W29, W30, W31
2	W21, W31	W27, W28, W29, W30
3	W29, W30	W21, W27, W28, W31

[1] Factory configuration

Note: DMA channel 1 is recommended, since IBM PCs use DMA channel 2 for floppy-disk drives, and channel 3 for hard-disk drives. The included PCI-20092S Software Drivers support only DMA channel 1.

2.8 Rate Generator

The Rate Generator can be used for pacing A/D conversions and DMA transfers. Its output is also available on pin 1 of connector P3. The Rate Generator is actually two concatenated 16-bit counters driven by an 8 MHz frequency reference. These

counters are referred to as the Rate Generator's Least Significant Word (LSW) and Most Significant Word (MSW) counters.

2.9 Installing the Board Into Your Computer

The procedure for installing your Board in an IBM PC is described below. The procedure for installing the Board in other machines may differ somewhat, but will be similar in nature.

1. Turn off the power to your computer, and unplug it.

> **WARNING**
> **Lethal voltages exist inside computers. Always ensure that power is removed before opening the case. Only qualified technicians should install, modify, or adjust equipment inside any computer unit.**

> **CAUTION**
> **Failing to turn off the power when inserting or removing boards will damage the boards and possibly the computer as well.**

2. Remove the five screws on the rear panel of the computer's chassis that hold the cover to the chassis.

3. Carefully slide the cover toward the front of the chassis, lift it up slightly, and remove it.

4. Select an empty expansion slot for your Board. You will be plugging the Board into this slot.

5. Remove the slot cover from the expansion slot.

6. Align the Board above its expansion slot connector, with the front (left) edge of the Board pushed against the card guide, and the card-edge fingers on the Board aligned with the connector in the expansion slot.

7. Gently push the Board down toward the connector, keeping the front (left) edge of the Board seated in the card guide. Press on the top of the Board at each end, and with a gentle rocking motion, insert the Board into the connector. *Do not force the Board, as this could break the connector.*

8. Attach the Board firmly to the computer, using the mounting screw on the bracket.

9. Connect the input/output cables to the Board.

10. Replace the computer's cover, using the five screws previously removed.

2.10 Pins and Ports: Connecting to the Outside World

2.10.1 ANALOG CONNECTOR (P2)

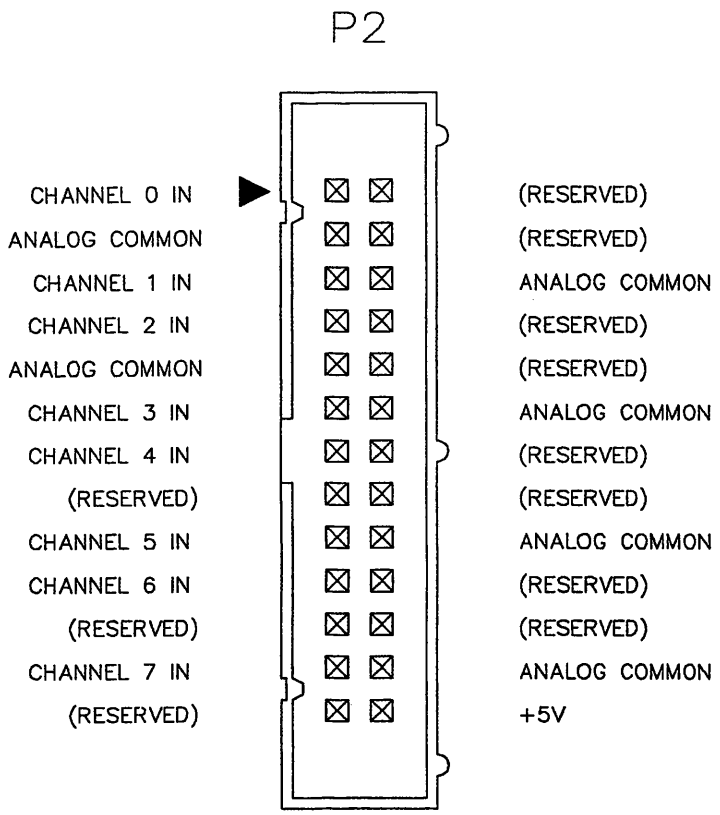
All analog signal connections are made through a 26-pin right-angle shrouded header protruding through the bracket. This connector (figure 5a) is compatible with our cables and Termination Panels. The connector is keyed to prevent reverse orientation.

2.10.2 DIGITAL CONNECTOR (P3)

A seven-pin digital I/O connector (figure 5b) transfers control signals between the PCI-20091W-1 and the "outside world." This connector is accessible through the bracket slot. The connector is keyed to prevent reverse orientation. The pin-out and signal description are given in table 2.9.

TABLE 2.9 Digital Connector (P3) Pinouts and Signal Description

Pin Number	Signal Description
P3 - 1	Rate Generator Output. This signal is the TTL-buffered onboard pacer clock. For a timing description see section 6.6, **CNF.RG** instruction.
P3 - 2	External Start Convert. This TTL level input initiates an A/D conversion cycle on a low-to-high edge. The Analog Input section must be properly jumpered for this option (see section 2.4).
P3 - 3	Convert Inhibit. Taking this pin to a TTL low level will inhibit any further A/D conversions from occurring. This signal does not affect software start convert commands.
P3 - 5	External Interrupt. This signal can be connected to generate a PC interrupt when taken to a TTL low level (see section 2.6). In DMA Mode 3, it can generate the DMA Trigger on a high to low transition. This signal is also available as a bit in the PCI-20091W-1 Status Register, which may be polled by a software.
P3 - 6	Digital Ground. Used for digital return on P3.
P3 - 7	+5 Volt Source. This source is current-limited at 250 mA (in addition to the limitation on Connector P2).



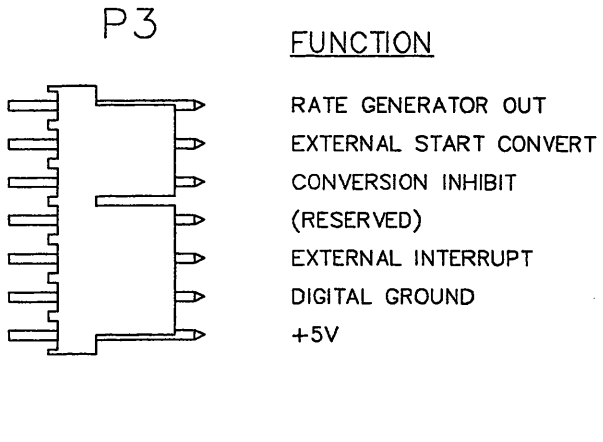


Fig. 2.4 Connector pin assignments a) Analog and b) Digital

CHAPTER 3: SOFTWARE INSTALLATION

3.1 Chapter Overview

This chapter covers installation of the PCI-20092S Software Drivers. Chapters 4 and 5 cover the topics of high speed analog data acquisition, Direct Memory Access (DMA), and buffer management in detail. The software technical reference chapter (chapter 6) gives information on creating application programs, lists and describes the calls used by the Software Drivers, and lists the error codes you may encounter. Sections 3.2 through 3.4 introduce the Software Drivers and list the hardware and software needed to run them. Section 3.5 gives instructions on how to make a working copy of your Software Drivers. Section 3.6 describes how the Software Drivers work. Section 3.7 tells you how to install your software into your computer, and how to get your software up and running.

3.2 Software Drivers Included With Your System

The PCI-20092S-1 Software Drivers, included with your PCI-20091W-1 Board, provide an uncomplicated, consistent, and useful interface between the Microsoft QuickBASIC language programmer and the PCI-20091W-1. They provide QuickBASIC support of the most useful hardware-level functions, while buffering the programmer from the details of running the hardware. We believe you will find the PCI-20092S-1 Software Drivers to be a very flexible, powerful, and useful personal computer interface between you and the PCI-20091W-1 High Speed Analog Input Board.

The PCI-20092S-1 Software Drivers are a prebuilt subset of the PCI-20026S and PCI-20027S Software Drivers. As such, they support the Microsoft QuickBASIC Language and most of the functions of your Board, including a high speed data acquisition mode and DMA data acquisition modes. The calls used in the PCI-20092S Software Drivers are identical to calls used in the PCI-20026S and PCI-20027S Software Drivers. The PCI-20026S and PCI-20027S Software Drivers offer greater capabilities in several areas. They support other languages (IBM Interpreted BASIC, C, and Turbo Pascal), other hardware in the PCI-20000 System, and a greater diversity of the high speed and DMA data acquisition methods.

As an illustration of the simplicity of the PCI-20092S-1 Software Drivers, consider the following example:

```
:
:
:
CALLS ABSOLUTE (CHN, GAIN, Z.CHN, RANGE, CNF.AI)
:
:
CALLS ABSOLUTE (AI.T, CHN, STOR, READ.CH)
:
:
:
```

The above two program lines show calls which first configure (prepare) an analog input channel to read data, and then read the analog input data from the specified channel. These calls, supported by the PCI-20092S-1 Software Drivers, allow you to interface with the hardware in a straightforward and painless way.

The variable names we use in our program listings are arbitrary, for the purpose of clarity. You may use any variable names which do not conflict with names defined by the PCI-20092S-1 software, or with the requirements of your programming language. (See figure 3.2, PCIHEADQ.BAS.)

3.3 System Configuration Requirements

Your personal computer system must use PC-DOS or MS-DOS version 2.0 or higher, and have approximately 33 Kbytes of memory for the PCI-20092S-1 Software Drivers, in addition to the memory required for the computer's operating system and QuickBASIC. One user-defined interrupt in the range 60 (Hex) through 67 (Hex) must be available (default value is 60 (Hex)). You can find more information on interrupt vectors, and how to change them, elsewhere in this manual.

In order to use the PCI-20092S-1 Software Drivers, Microsoft QuickBASIC must be available on the computer.

3.4 The PCI-20092S Software Drivers License Agreement

Read the PROGRAM LICENSE AGREEMENT, printed on the envelope containing the PCI-20092S-1 Software Drivers diskette, and make sure you fully understand it. If you have any questions, contact your sales representative before continuing.

3.5 Creating a Working Copy of Your Software

The PCI-20092S-1 Software Drivers system diskette, which is for Microsoft QuickBASIC language programmers, contains the following files:

SOFTWARE DRIVERS GROUP:

PCI_92S.EXE Software Drivers program
PCIHEADQ.BAS Header file

SAMPLE PROGRAMS:

SAMPLE01.BAS
SAMPLE02.BAS
SAMPLE03.BAS
SAMPLE04.BAS
SAMPLE05.BAS

GRAPHICS DEMONSTRATION:

GRAPH.BAS

OPTIONAL TEXT FILE:

README

Use of the Software Drivers is discussed in the remaining sections of this chapter. The Sample Programs are listed in appendix A. The optional text file, if present, contains information that is too recent to appear in this manual, and should be read. It may be read by typing MORE< README at the DOS prompt. (MORE is a DOS external command, meaning that it is a file on the DOS diskettes.)

Before you begin to experiment with your Software Drivers, you should make a working copy of your master diskette, containing a copy of all of the files on your original diskette. You should **never** use the original diskette for everyday operations. Once you have made this working copy, use it for your everyday operations, and store your master diskette in a safe place.

3.5.1 PROCEDURE FOR ONE OR TWO FLOPPY DISK DRIVES

Place your DOS operating system diskette in drive A: and type:

```
DISKCOPY A: B:
```

and follow the instructions that appear on the screen to make your working copy. This command tells DOS to copy all the files from the diskette in your A: drive to the diskette in your B: drive. If you wish to check the integrity of your backup when the process completes, place your DOS diskette in drive A: again and type

```
DISKCOMP A: B:
```

to verify that there are no comparison errors. Note that DISKCOPY and DISKCOMP are both DOS "external" commands, meaning they are files on the PC-DOS

(or MS-DOS) system diskettes. If you have only one floppy disk drive, the same commands will work; just follow the instructions on the screen.

After you have copied your software drivers diskette, store your original diskette in a safe place, and use the working copy for all subsequent operations.

3.5.2 PROCEDURE FOR ONE FLOPPY DISK DRIVE AND A FIXED DISK DRIVE

If you have a fixed (hard) disk drive and only one floppy disk drive, you can create a backup while installing the software on your hard disk. First create a subdirectory by typing the DOS command

```
MD subdirectory
```

This tells DOS to Make a Directory with the name you type in after MD. Note that the restrictions on file names also apply to directory names: they can only be eight characters long, with an optional three-character extension. See your DOS manual for further details.

After you have created a subdirectory, place your master diskette in your floppy disk drive (the A: drive), and type at the DOS prompt

```
COPY A:*.* C:\subdirectory/V
```

This tells DOS to copy all the files from your A: drive to your subdirectory, verifying that each file is copied correctly. To double-check the integrity of your files, you might wish to type

```
COMP A:*.* C:\subdirectory
```

This command will cause DOS to compare the files on your master diskette with the files in your subdirectory. Note that COMP is a DOS "external" commands, meaning it is a file on the PC-DOS (or MS-DOS) system diskettes.

Now that you have created your subdirectory, and have copied all the files from your master diskette to that subdirectory, you can either treat that subdirectory as your working "diskette," or you can copy those files onto a second floppy diskette, and use that second floppy as your working backup. If you wish to use the subdirectory, stop here and skip to section 3.6. If you wish to make a separate floppy diskette into your backup, place a formatted, blank diskette in the A: drive, and type

```
COPY C:\subdirectory\*.* A:/V
```

To verify the integrity of your files, type

```
COMP C:\subdirectory\*.* A:
```

Once you are sure you have a complete copy of your master diskette, put your master diskette away and use the diskette you have just created as your working copy.

3.6 Software Drivers Description

The structure of the PCI-20092S-1 Software Drivers system is shown schematically in figure 3.1. The PCI_92S.EXE program is the controlling center of the system. The user loads it into memory, where it remains resident for use by QuickBASIC programs.

When a PCI-20092S-1 Software Drivers instruction is called, program control is passed from QuickBASIC to the PCI_92S.EXE routines, as shown in figure 3.1. All of the following steps are then taken care of by the Software Drivers routines, and are completely transparent to the user. The computer jumps to the appropriate routines in PCI_92S.EXE to execute the desired function. These routines maintain a local data area, the *System Configuration Data Area*, which is contained within the resident program area, and which keeps track of the current state of the hardware. The data area is initialized by the **SYSINIT (System Initialize)** function, and local data structures are built by the **INIT (Initialize)** function. These structures are modified by "Configuration" functions. I/O functions such as **READ.CH (Read Channel)** and **WRITE.CH (Write Channel)** make use of the stored data, and access the *Hardware Drivers* to perform the desired function.

In the event that an error results from an instruction call, the Software Drivers will return to the user's Microsoft QuickBASIC program with an error code set in the system error-code register. The contents of the system error-code register can be read by executing the **ERR.SYS (System Error)** instruction. To interpret the meaning of the error code returned, refer to sections 6.7 through 6.13.

When a PCI-20092S-1 instruction is called, the call is actually handled by the QuickBASIC ABSOLUTE utility, which uses its final argument to locate the offset of the instruction's entry point in the PCI_92S.EXE jump table. These entry-point offsets are defined in the header file PCIHEADQ.BAS. Figure 3.2 shows a typical header file.

PCIHEADQ.BAS also defines mnemonics for the various I/O Type codes, which are required by the **READ.CH (Read Channel)** and **WRITE.CH (Write Channel)** instructions.

Note that the entry-point offsets and I/O-type constants ***must*** be QuickBASIC integers. This is ensured in line 2, which defines all variables except those beginning with **F** to be integers; we have reserved **F** for floating-point data conversions. Therefore, be sure to avoid variable names that start with **F** unless they relate to floating-point data conversions.

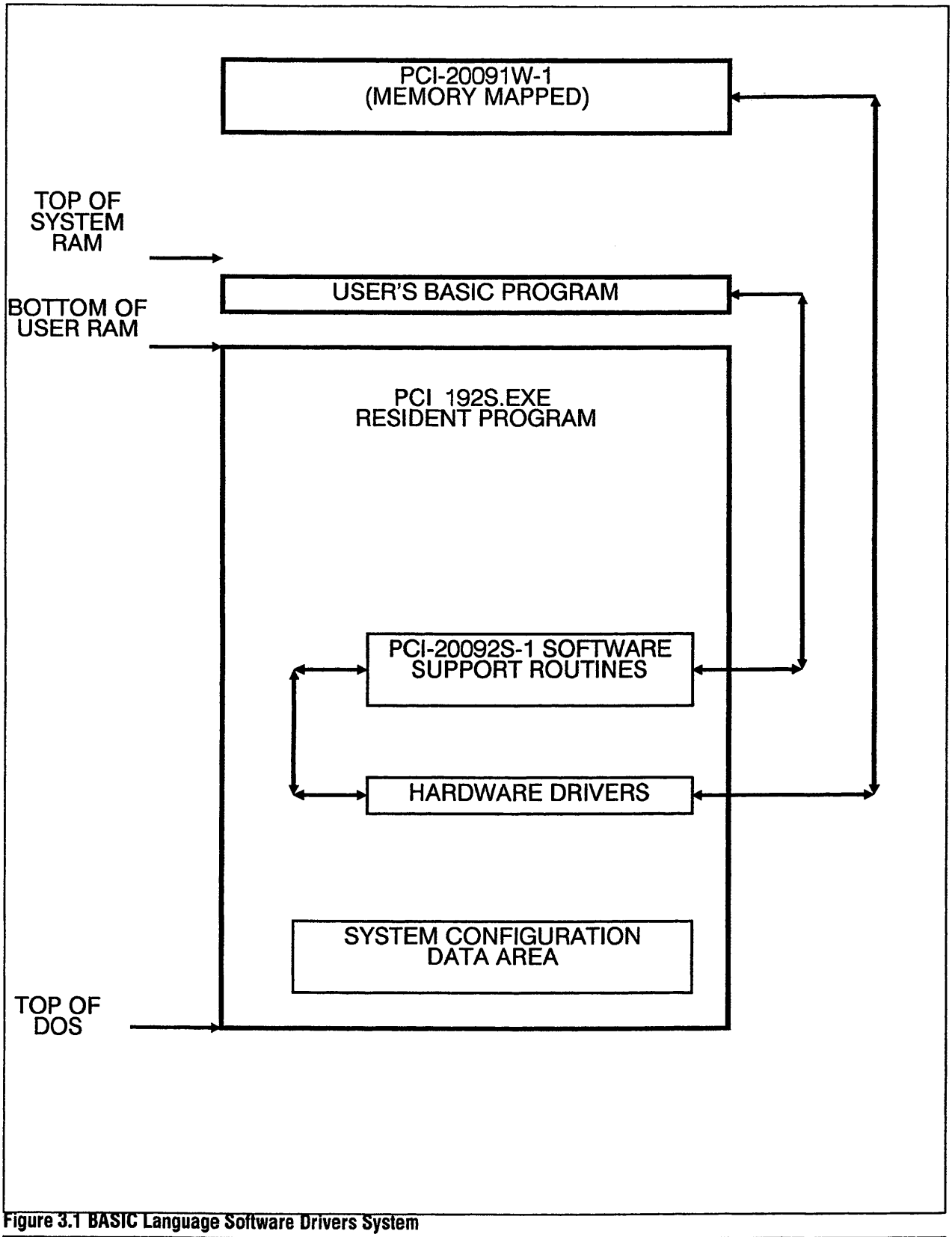


Figure 3.1 BASIC Language Software Drivers System

```
1   ' *** Header File for PCI-20092S-1 Microsoft QuickBASIC Applications ***
2   DEFINT A-E, G-Z
3   ' Define the internal signature
4   CONST VCHK.L = 15, VERS.L = 15
5   DIM VCHK (VCHK.L), VERS (VERS.L)
6      VERS (1) = 257
7      VERS (2) = 17
8      VERS (3) = 17
9      VERS (4) = 0
10     VERS (5) = 0
11     VERS (6) = 17
12     VERS (7) = 4352
13     VERS (8) = 0
14     VERS (9) = 4096
15     VERS (10) = 4369
16     VERS (11) = 0
17     VERS (12) = 0
18     VERS (13) = 0
19     VERS (14) = 4352
20     VERS (15) = 17
21  ' Define the PCI-20092S-1 instruction offsets.
22     CONST AUTOGRPH = 12
23     CONST BUF.DEC = 18
24     CONST CNF.DMA = 24
25     CONST CNF.HS = 30
26     CONST CNF.RG = 36
27     CONST DMA.RUN = 42
28     CONST DMA.STAT = 48
29     CONST DMA.STOP = 54
30     CONST ERR.SYS = 60
31     CONST HS.RUN = 66
32     CONST INIT = 72
33     CONST READ.CH = 78
34     CONST SYSINIT = 84
35     CONST WRITE.CH = 90
36  ' Define the PCI-20092S-1 I/O types.
37     CONST AI.T = 1
38     CONST RG.T = 8
39     CONST EXINT.T = 100
40     CONST XSYNC.T = 104
41     CONST AIX.T = 105
42  ' Buffer functions present.
43     CONST BUF = 0
44  ' DMA functions present.
45     CONST DMA = 0
46  ' High-speed acquisition functions present.
47     CONST HSA = 0
48  '
49  DEF SEG = 0 : INTVEC = &H180
50  SEGMT = PEEK (INTVEC+2) + &H100*PEEK (INTVEC+3)
51  DEF SEG = SEGMT
52  ' Check for installed software
53  IF (SEGMT <> 0) AND (PEEK (11) = 33) AND (PEEK (12) = &H9A) THEN 2
54  1 PRINT "PCI-20092S-1 not installed or invalid.", CHR$ (7) : END
55  2 CALLS ABSOLUTE (VERS.L, VCHK (1), VCHK.L, AUTOGRPH)
56  IF (VCHK.L <> VERS.L) THEN 1
57  FOR I = 1 TO VCHK.L : IF (VERS (I) <> VCHK (I)) THEN 1
58  NEXT I
```

Fig. 3.2 PCIHEADQ.BAS listing

The instructions in lines 49 through 58 verify the presence of the resident software, and set the segment used by call statements to reference the PCI_92S.EXE jump table.

The PCIHEADQ.BAS header file is stored in ASCII format. It may be incorporated into an application program with the $INCLUDE metacommand, as described in Chapter 5.

3.7 Loading the Software Drivers Routines

The PCI_92S.EXE Software Driver routines must be loaded before running any application program which calls them. The Software Drivers routines must be loaded from DOS before starting QuickBASIC. They cannot be loaded from QuickBASIC's DOS-command utility. If you create an application program, you will not be able to run it unless the Software Drivers routines were loaded before you began your session. If this was not done, you will have to save your work and exit to DOS, load PCI_92S.EXE, and then resume your work. If you compile your QuickBASIC application to an .EXE file to be run after you exit QuickBASIC, you do not have to load the Software Drivers until you are ready to run the application.

To load the PCI-20092S Software Drivers, place your work disk in the computer or select the directory on your fixed disk containing the Software Drivers files. At the DOS prompt, type

```
PCI_92S [/Va] [/G] [/X"text"]
```

then press the RETURN key. Actual examples are given in section 3.7.4, *EXAMPLES OF LOADING THE SOFTWARE DRIVERS*, below. The switches are described below, and are summarized in table 3.1.

You must have a copy of the DOS file COMMAND.COM in the drive you use to boot your system (normally A: or C:). PCI_92S.EXE loads a second copy of COMMAND.COM, which requires about 3-5 KBytes in addition to the memory allocation of the Software Drivers routines themselves. The Software Drivers will display a sign-on message. After the drivers have been successfully loaded, you will see a DOS sign-on message and a DOS prompt.

If the file COMMAND.COM cannot be found, or if other errors are encountered, you will see an error message after the PCI-20092S-1 sign-on messages. For more information on this and other error messages, see the discussion below.

The Software Drivers routines are automatically unloaded (along with the second copy of COMMAND.COM) when the DOS command EXIT is typed. You will see the message

```
PCI-20092S:  End of execution
```

when PCI_92S.EXE terminates. Note that if you run batch files which contain the EXIT command for the purposes of nesting batch files, you may inadvertently remove the resident Software Drivers routines.

3.7.1 ERRORS WHILE LOADING THE SOFTWARE DRIVERS

If an error (usually a command line error) is detected during preliminary processing, the software may display one or more of the following messages.

```
*** Invalid dispatch vector
*** COMSPEC not set in environment
*** Invalid execute specification
*** Memory allocation error
*** Unrecognized switches
```

The software drivers perform a timing calibration test on your computer during the load process. If this test fails, you may see one of the following messages.

```
System clock driver not running
```

OR

```
Interrupts detected on level(s) n....Retry failed.
```

In any of these conditions, the following will appear:

```
Errors detected; code not installed
```

If non-fatal command-line errors are detected, the following messages may appear:

```
*** Extra characters on input line
Input line errors, ignored
```

If the error is detected as the Software Drivers attempt to load the command processor (COMMAND.COM), the following message will be displayed

```
PCI-20092S: Execute failed
```

followed by one of these informative messages

```
Invalid function
File not found
Memory allocation error or memory control blocks damaged
Insufficient memory
Incorrect segment
Bad environment
Bad format
Undefined error
```

In the case the error "File not found," a message similar to the following will appear.

```
Can't find C:\COMMAND.COM
Insert boot disk and press <ENTER> to retry.
```

Place your DOS diskette containing COMMAND.COM in the drive you use to boot your system, and press the <ENTER> key. When the <ENTER> key is pressed, the Software Drivers will try again to load the command processor. If any other key is pressed, the Software Drivers will exit.

3.7.2 MEMORY RESIDENT PROGRAMS

If you use memory resident programs (also known as TSRs) such as Borland's Sidekick or SuperKey, or DOS external commands PRINT or MODE, you should load these programs *before* you load the PCI-20092S-1 software. If you load memory resident programs after loading PCI-20092S-1, be sure to unload them, in the order they were loaded, before unloading PCI-20092S-1. (Some memory resident programs, including PRINT and MODE, cannot be unloaded.)

If you use DOS environment commands such as PATH, PROMPT, and SET, you should give these commands before loading any memory resident programs; otherwise you may get the DOS error message "out of environment space." The environment size can be increased for DOS versions 3.2 and later; consult your DOS manuals if you need to do this.

The precautions noted above for memory resident programs also apply if you use a "windowing" program such as TopView. In addition, you should never have two applications, both of which access the same copy of PCI-20092S-1 routines, running simultaneously. Running PCI-20092S-1 in conjunction with a windowing program is not recommended, and is not guaranteed to work.

If you use a shell program to replace or supplement the DOS command interpreter, you may wish to load the Software Drivers before you enter the shell program. If you load the Software Drivers from the shell program, you will not return to the shell until you exit the Software Drivers.

Device Drivers are a special type of memory resident program which are loaded automatically during your computer startup sequence. These files usually have the extension SYS. Some device drivers and memory-resident programs enable interrupts from an auxiliary device or intercept one or more system interrupts, which may reduce the system performance. For this reason, you may wish to avoid using any unnecessary memory resident programs or device drivers when you are acquiring data with PCI-20092S-1.

3.7.3 SWITCHES

SWITCH /V (INTERRUPT VECTOR)

Interrupt vector 60 (Hex) is used to store the beginning address of the resident routines. The switch /V, which is used when invoking PCI_92S.EXE, determines the interrupt vector to be used in calling the Software Drivers.

This switch, in the form **/Va** (where **a** is a variable), may be used to specify an interrupt vector other than the default interrupt vector of 60 (Hex). PC DOS has reserved vectors 60 through 67 (Hex) for definition by the user. If you specify any vector other than the default value of 60 (Hex), you must make sure that the vector will be recognized by your applications programs. The method of accomplishing this is given in the section on creating application programs. Note that in the IBM PC, interrupts are allocated four bytes, so each interrupt vector is calculated to be at an address four times the interrupt number. However, if **a** is greater than FF (Hex), it is interpreted as an offset in Segment 0000, rather than as an interrupt vector, and the automatic multiplication by 4 will not be performed. For example, **/V180** is equivalent to **/V60**.

You will only need to use switch /V when loading PCI_92S.EXE if you have another application that uses interrupt 60.

Note: If you use any vector other than the default of 60 (Hex), you must change the reference to this vector in the header to your program. In the header file PCIHEADQ.BAS, which you copied to your work disk, this vector is specified in the line defining "INTVEC" (in the typical PCIHEADQ.BAS, figure 3.2).

Note on Expanded Memory: Expanded memory drivers use interrupt 67 (Hex).

SWITCH /G (BYPASSING CONFIRMATION)

Normally, PCI-20092S-1 stops after displaying the sign-on message to request user confirmation. The switch /G causes the Software Drivers to "Go", bypassing user confirmation as it is loaded. If you run PCI_92S.EXE from a batch file, you will probably use this switch.

SWITCH /B (ALLOCATE DMA BUFFER)

When the PCI-20092S Software Drivers are loaded into memory, a buffer may be allocated by specifying the **/Bb** load option switch, where **b** is an integer variable. This switch allocates a buffer at the "top" of memory. If the **b** parameter is not present, a default buffer 64 Kbytes in length will be allocated. If **b** is present, **b** Kbytes will be allocated. Any size buffer from 1 to 64 Kbytes may be allocated.

SWITCH /X (EXECUTE COMMAND LINE)

The switch /X"**text**" allows the user to specify a command line to be executed by PCI_92S.EXE. The command line **text** can be any command line you would type at the DOS prompt. It may invoke any batch file or any executable (EXE or COM) file. Command-line arguments may be included. **Note:** You must type the double quotes, or an acceptable substitute, around the text string. Acceptable substitutes are single quotes (/X'**text**') or square brackets (/X[**text**]). The characters that you choose should be ones that do not appear within the text string. *If the command line contains DOS pipe or redirection characters (|, <, or >), you must use double quotes.*

The /X switch is useful for applications that are designed to run automatically. If PCI_92S.EXE is invoked from a batch file, it should appear on the last line of that batch file. You can then continue batch processing by specifying a second batch file with the /X switch.

TABLE 3.1 Summary of PCI_92S.EXE Switches

Switch	Description
/Va	Allows the user to specify an interrupt vector other than the default value of Hex 60.
/G	Bypasses user confirmation. Useful for batch file operations.
/Bb	Allocates a DMA buffer **b** Kbytes in length. Allowable values for **b** are 1 to 64 Kbytes.
/X"text"	Specifies a command line to be executed by PCI_92S.EXE.

3.7.4 EXAMPLES OF LOADING THE SOFTWARE DRIVERS

If you are loading PCI_92S.EXE from a batch file, you must put the command

```
PCI_92S switches
```

as the last line of the batch file. Lines after the PCI-20092S-1 command will not be processed until PCI_92S.EXE exits and returns control to the original copy of COMMAND.COM.

If you want to continue batch processing, you can have PCI_92S.EXE run a second batch file (or any executable file). To do this, you must specify the file to be loaded using the /X load option switch.

Example 1: The following batch file calls PCI_92S.EXE, allocates a 32K DMA buffer, starts QuickBASIC.

```
PCI_92S/G/B32/X"QB /L"
```

Example 2: The following batch file (which might be an AUTOEXEC.BAT file) sets up some operating system parameters, calls PCI_92S.EXE, and invokes a second batch file. It also invokes internal DOS commands CLS, PATH, and PROMPT.

```
CLS
PATH C:\;C:\WORK
PROMPT $p$g
PCI_92S/G/X"AUTO2 DATA1"
```

The second batch file, AUTO2.BAT, runs two programs and saves a data file:

```
PROG1 %1
COPY %1 C:
PROG2
```

(The %1 symbol passes the parameter "DATA1" to the batch file. Batch-file processing is discussed in detail in your DOS manual.)

Note that when the batch files or programs specified by the **/X** switch have finished running, and the computer returns to DOS, the PCI-20092S-1 routines will automatically unload.

CHAPTER 4: HIGH SPEED ANALOG DATA ACQUISITION

4.1 Introduction and Chapter Overview

This chapter contains an in-depth discussion of the concepts related to High Speed analog acquisition, and how the PCI-20092S-1 Software Drivers operate to accomplish this type of data acquisition. Section 4.3 gives high speed data acquisition programming guidelines; section 4.4 describes the High Speed data acquisition mode.

Many applications require the acquisition of a high speed "snapshot" of analog data. The snapshot may be initiated by some sort of trigger, after which the analog signals are sampled at accurately timed intervals. The PCI-20091W-1 is very well suited to applications of this kind. You may select one of two time-base signals and which channels to read. Acquisition rates of up to 89,000 samples per second may be achieved using high speed acquisition with the PCI-20091W-1.

The PCI-20027S Software Drivers support three different high speed analog data acquisition modes with the PCI-20091W-1 Board. As a subset of the PCI-20027S Software Drivers, the PCI-20092S Software Drivers support High Speed Mode 4. This mode provides the highest speed acquisition with hardware controlled channel selection and conversion timing. (High speed modes supported by PCI-20027S are Mode 1, with software-controlled timing and more flexible timing options, and Mode 3, with software-controlled channel selection.)

High Speed analog data acquisition can be performed with a single PCI-20091W-1 Board and the PCI-20092S-1 Software Drivers. The PCI-20091W-1 must be configured to provide a source of clock signals to pace the acquisition, and may be configured to provide a gate which will determine when the acquisition will begin.

4.2 Hardware Configuration

This section describes the hardware configuration for a high speed data acquisition. You will need to select a time base, and you may optionally use a trigger source and automatic channel selection.

The time base for the high speed acquisition may be provided by the PCI-20091W-1's onboard rate generator, or by an external start-convert signal. In either case, you must configure the Board with appropriate jumpers. To use the external Hardware Start Convert, you must provide an external signal (TTL-compatible) at the appropriate pin of the PCI-20091W-1 digital connector. Refer to section 2.4 for the appropriate jumper configuration in either case.

You may provide a signal to the Conversion Inhibit input in order to trigger data acquisition to begin following a desired event. When the the Conversion Inhibit input goes high, data acquisition will commence. If analog conversions fail to take place, either because Conversion Inhibit remains low, because pacing signals are absent, or because the pacing signals are not reaching the A/D converter, the computer will effectively be locked up, necessitating a system reset.

If your application involves sampling a single channel, you can either use channel 7 for your input signal or disable the automatic channel sequencing capability of the Board. If your application involves sampling more than one channel, you must connect your input signals to channels **N** through 7, where **N** depends on the number of input signals. You must enable automatic channel sequencing. Refer to section 2.5.

4.3 Programming Guidelines

The instructions used for high speed data acquisition are listed and commented in table 4.1 in the order you would call each instruction. Section 4.4 describes High Speed Mode 4; chapter 6 describes each individual instruction in greater detail. Not all the instructions given in table 4.1 will be used for every data-acquisition configuration. Be sure to refer to the entry for each instruction in chapter 6 for full details on the instruction arguments.

TABLE 4.1 High Speed Programming Instructions

CNF.RG	If the PCI-20091W-1 rate generator is used as a time base, use this instruction to set up the rate generator channel. **CNF.RG**, if used, must be called once before the first call to **HS.RUN**. This instruction is not required if you are using the external start convert input as the acquisition pacer.
CNF.HS	Use this instruction to specify the time base of the acquisition, High Speed Mode 4, and the list of analog channels that will be read during the acquisition. **CNF.HS** must be called once before the first call to **HS.RUN**.
HS.RUN	Use this instruction to perform the high speed data acquisition. Specify the number of passes through the channel list, and the address of the array where the data will be stored.
ERR.SYS	If an error occurs, this instruction will provide information as to exactly where in the program the error occurred, and the nature of the error. We recommend that you call this instruction after every call.

4.3.1 TIMING

If the onboard rate generator is used for timing, the data acquisition sampling rate is determined by the rate generator counts **n1** and **n2** in the **CNF.RG** instruction.

You may provide a signal to the Conversion Inhibit input in order to trigger data acquisition to begin following a desired event. When the the Conversion Inhibit input goes high, data acquisition will commence. If analog conversions fail to take place, either because Conversion Inhibit remains low, because pacing signals are absent, or because the pacing signals are not reaching the A/D converter, the computer will effectively be locked up, necessitating a system reset.

If your application involves sampling a single channel, you can either use channel 7 for your input signal or disable the automatic channel sequencing capability of the Board. If your application involves sampling more than one channel, you must connect your input signals to channels **N** through 7, where **N** depends on the number of input signals. You must enable automatic channel sequencing. Refer to section 2.5.

4.3 Programming Guidelines

The instructions used for high speed data acquisition are listed and commented in table 4.1 in the order you would call each instruction. Section 4.4 describes High Speed Mode 4; chapter 6 describes each individual instruction in greater detail. Not all the instructions given in table 4.1 will be used for every data-acquisition configuration. Be sure to refer to the entry for each instruction in chapter 6 for full details on the instruction arguments.

TABLE 4.1 High Speed Programming Instructions

CNF.RG	If the PCI-20091W-1 rate generator is used as a time base, use this instruction to set up the rate generator channel. **CNF.RG**, if used, must be called once before the first call to **HS.RUN**. This instruction is not required if you are using the external start convert input as the acquisition pacer.
CNF.HS	Use this instruction to specify the time base of the acquisition, High Speed Mode 4, and the list of analog channels that will be read during the acquisition. **CNF.HS** must be called once before the first call to **HS.RUN**.
HS.RUN	Use this instruction to perform the high speed data acquisition. Specify the number of passes through the channel list, and the address of the array where the data will be stored.
ERR.SYS	If an error occurs, this instruction will provide information as to exactly where in the program the error occurred, and the nature of the error. We recommend that you call this instruction after every call.

4.3.1 TIMING

If the onboard rate generator is used for timing, the data acquisition sampling rate is determined by the rate generator counts **n1** and **n2** in the **CNF.RG** instruction.

Refer to the discussion of the **CNFRG** instruction in chapter 6 for details on setting this frequency.

Tests performed using an IBM PC with an 8088 microprocessor and an internal clock speed of 4.77 MHz, and an IBM AT with an 80286 microprocessor and an internal clock speed of 6 MHz, resulted in sampling rates of 62,500 samples/ second and 89,000 samples per second respectively. Faster times are possible if you use a computer with a faster internal clock, a faster microprocessor, or both.

4.3.2 TESTING HIGH SPEED SAMPLING RATES

These paragraphs provide a suggested test procedure for determining the maximum high speed data acquisition rate possible for your configuration. More information on calculating the sampling rate can be found in chapter 6 under the description of the **CNFRG** instruction.

Connect input channels 0 through 7 of the PCI-20091W-1 to a known static input pattern. For example:

Channel 0: 0 volts

Channel 1: 1 volt

:

:

Channel 7: 7 volts

Jumper the PCI-20091W-1 for automatic channel advance (see section 2.5 for information on setting jumpers for channel advance).

Perform a high speed acquisition with the channel list consisting of the above channels. Examine the data for several runs of several passes each to verify that correct data is being read.

4.4 High Speed Mode 4: Hardware-Controlled High Speed Analog Acquisition

4.4.1 DESCRIPTION

High Speed Mode 4 is a data acquisition mode with completely hardware-controlled timing and channel selection. This mode is fast and accurately timed.

A timing signal is supplied to the start convert input of the A/D converter, and the same signal controls automatic channel selection. A trigger event may be used to gate the timing signal input to the start convert input, or a gate applied to the Convert Inhibit input. Once the acquisition is under way, the software will read data from the proper analog input channel(s) each time the end of conversion signal from the PCI-20091W-1 is detected.

When data acquisition takes place with hardware controlled timing and channel selection, the Software Drivers perform the following steps:

1. When conversion is complete, read and store the data.
2. Check the number of samples taken, and go to step (1) if sampling is not complete.

4.4.2 PROGRAMMING

See the discussion in section 4.3 for information on high speed data acquisition programming.

CHAPTER 5: DMA DATA ACQUISITION AND CONTROL

5.1 Introduction and Chapter Overview

This chapter describes how the PCI-20092S-1 Software Drivers operate direct memory access (DMA). This section contains an introduction to the PCI-20092S-1 Software Drivers DMA processes, including a discussion which should help you determine at a high level whether DMA is appropriate to your needs.

The PCI-20027S Software Drivers support nine different DMA modes. The PCI-20092S Software Drivers, as a subset of the PCI-20027S Software Drivers, support three of those modes: DMA Mode 2, DMA Mode 3, and DMA Mode 4. These modes are described in sections 5.2, 5.3, and 5.4.

DMA data acquisition rates are discussed in section 5.5. Operating other programs while DMA is in progress is described in section 5.6. Storage of DMA data is discussed in sections 5.7 and 5.8. Programming DMA data acquisitions using the included Software Drivers is described in sections 5.9 and 5.10.

5.1.1 DMA INTRODUCTION

The PCI-20092S-1 Software Drivers DMA makes the full power and flexibility of the PCI-20091W-1 available. DMA is a hardware technique that allows you to take data without software intervention. This gives you high speed data transfer and true background operation. While DMA is running, the only computer resources it uses are the memory it is filling up, one channel of your computer's DMA controller, and the bus cycles necessary to complete the transfers. Data transfer speeds up to 89,000 samples per second can be achieved on an IBM XT.

The PCI-20092S-1 DMA process uses data buffers (locations in memory used to store input data while the DMA process is taking place). You can allocate memory for a data buffer when the PCI-20092S-1 drivers are loaded by setting a switch on the command line. See section 3.7 for details on switches.

5.1.2 SEQUENCE OF OPERATION

The PCI-20092S-1 Software Drivers sets up the PCI-20091W-1 and the host PC to perform a DMA process. The hardware then performs the data acquisition process. While the DMA process is running in the background, control returns to the user's application program, and other processing may occur in the foreground. The application program can query the Software Drivers for the status of the DMA process and monitor the data being acquired. When the process terminates, the application program can process the acquired data.

5.1.3 INFORMATION REQUIRED FOR RUNNING DMA

Once the decision has been made to use a DMA process for data acquisition, the following questions should be considered:

1. What data is to be transferred by DMA? The PCI-20091W-1 can transfer data from one to seven analog input channels. The desired inputs must be connected to appropriate channels, and automatic channel sequencing must be enabled if appropriate.

2. How much data is to be transferred? An adequate data buffer must be available, and the software must be programmed accordingly.

3. How is the DMA process to stop? The DMA process can acquire a specific amount of data, or it can terminate when a specific event has been recorded. This determines the DMA mode to be used. A signal used to stop the DMA process is called the **DMA Trigger Event.**

4. What signal is to be used to pace the transfer of data? The PCI-20091W-1 Board uses the A/D converter's End of Conversion signal to control the DMA process. This signal is the **DMA Source Event.** The A/D conversion rate is in turn determined by the signal connected to the Start Convert input when you configure your hardware system.

5. What signals must be connected in order to ensure that all components of the system function automatically during the data acquisition? Since DMA is a hardware technique, certain jumpers must be installed on the PCI-20091W-1 in order to make the proper signals appear in the right places. Refer to chapter 2 for jumper configuration details.

The answers to these questions provide the PCI-20091W-1 hardware with the parameters it requires for a successful DMA operation.

5.2 DMA Mode 2. Start on Command; Stop on Terminal Count

DMA Mode 2 is a fixed-period data acquisition mode. The DMA process starts with a software command, and terminates after a specified amount of data has been transferred to the host computer. Use this mode to sample continuous input signals for a definite length of time.

5.3 DMA Mode 3. Start on Command; Stop on Trigger after Delay

In DMA Mode 3, the DMA process starts transferring data to the host computer on a software command, and continues to do so until a specified trigger event is received. Then, the DMA process continues for a specified number of data acquisi-

tions, and stops. Only the latest data are stored in the buffer; earlier data is overwritten after the data buffer has been filled once.

This DMA Mode can leave both pretrigger and posttrigger data in the buffer. The location of the trigger point in the buffer may be calculated by counting back from the end of the buffer by the length of the programmed delay. All data before the trigger point is pretrigger, and all data after this point is posttrigger. (The PCI-20092S-1 Software Drivers always put the oldest data in the beginning of the buffer.)

5.4 DMA Mode 4. Start on Command; Stop on Command.

DMA mode 4 is a continuous DMA process. The DMA process is started on computer command and Analog Input data from the PCI-20091W begins filling the DMA buffer. When the end of the buffer is reached, the computer's DMA controller resets to the beginning of the buffer and begins overwriting old data with new. The process stops on computer command, leaving only the most recent samples in memory. The number of samples collected depends on the size of the buffer used.

5.5 DMA Rates

The maximum rate (speed) at which a DMA process will run without overrun errors is dependent on several factors. The rate is defined as the average number of bytes per second at which data is transferred by the DMA process. For example, if you have set up a DMA rate of 20,000 samples per second, then the average byte-per-second rate is 40,000. The absolute maximum DMA rate with this system is 89,000 samples per second. The factors affecting maximum DMA speed include:

1. The amount of data being acquired. The smaller the burst of data, the faster it can be sampled.

2. The host computer being used. Variations in maximum DMA rates occur for different PC compatibles. IBM ATs and compatibles are actually slower than PCs because of the way DMA is implemented internally.

3. The nature of the foreground program. If the processor is executing instructions which use many bus cycles and will not relinquish the bus, then the maximum DMA rate will be much lower. If your program outputs to the screen and causes it to scroll, or clears the screen, then you may get an overrun error if a DMA process is running at the same time.

If your program sets up a DMA process which is too fast for your system, the DMA controller will do its best to execute it. However, if a pacing signal occurs before the previous data word is completely transferred, that pacing signal is ignored and the corresponding data is lost, and the overrun error flag is set on the PCI-20091W-1. This does not stop the DMA process; the next pacing signal will be honored, if possible. The data, however, will have gaps.

5.6 DMA and Foreground Programs

The DMA process operates by "stealing" bus cycles from the processor. The programmer needs to be aware that even though a DMA process is running in the background, while it is running it will affect the execution speed of any foreground program. The faster a DMA process is running, the slower the foreground program will run. DMA Mode 2 will typically not affect foreground program execution since processes in this mode normally don't last long enough. DMA Modes 3 and 4, however, can have a dramatic (but predictable) effect on foreground program execution.

The execution speed of the foreground program is affected by the DMA rate (the average number of bytes per second being transferred).

Table 5.1 shows foreground-program speed degradation versus byte-per-second DMA transfer rate for an IBM XT. This table may be used as a guide. We recommend that you test for a maximum DMA transfer rate using the conditions under which you intend to run your application. This may be accomplished by setting up a DMA run, and then monitoring the overrun flag in the host computer. Increase the rate of DMA transfers until the flag becomes true. The overrun condition will set an error code when the **BUF.DEC (Decode Buffer)**, **DMA.STAT (DMA Status)**, or **DMA.STOP (Stop DMA)** instructions are called. Use the **ERR.SYS (System Error)** instruction to check for the overrun error code.

TABLE 5.1 Effect of the DMA Process on Foreground Programs

DMA Transfer Rate in Ksamples per Second	Percent Degradation of: Foreground Program Execution Speed
10	4
50	15
70	23
89	30

Because of the effect of the DMA process on foreground program execution, you must use caution if your foreground program is performing time-critical tasks, such as high speed data acquisition.

5.7 DMA Buffers

A data buffer is a block of program memory used for temporary storage of input data during data acquisition. The PCI-20092S-1 Software Drivers store data in an input buffer during a DMA data acquisition process. When the process is complete, the

application program can read the data from the buffer and process, display, or store it. Details on loading buffers are in given in section 2.6.

5.7.1 BUFFER SIZE

Because of the internal design of IBM PCs and compatibles, a DMA process cannot automatically transfer data into or out of a buffer which crosses a 64-Kbyte page boundary. (A 64-Kbyte page boundary is a memory address evenly divisible by 65536 (10000 Hex). The 640 Kbyte DOS memory space comprises ten pages.) For this reason, the DMA Modes on this Board are limited to a maximum of 64 Kbytes of data.

5.7.2 BUFFER DATA ORGANIZATION

The logical structure of a buffer may be either "linear" or "circular," depending on which DMA Mode you specify in the **CNF.DMA (Configure DMA)** instruction.

DMA Mode 2 uses a simple **linear** buffer. This means that the DMA process starts loading data into the buffer at the buffer's lowest address, and continues to load data into the buffer until the buffer is full or the DMA process reaches the programmed number of bytes (terminal count), whichever comes first. At that point, the DMA process stops. Data is stored sequentially in the buffer, and is always left-justified in the buffer. That is, the oldest data is at the beginning of the buffer, and the newest data is at the end.

DMA Modes 3 and 4 use what is called a **circular** buffer. Data is stored sequentially in the buffer, as above, but if the DMA process reaches the end of the buffer, it starts over again at (wraps around to) the beginning, until the stopping criterion is met. Newly acquired data will thus overwrite the old data, and when data acquisition stops, the oldest data may lie somewhere in the middle of the buffer. The PCI-20092S-1 software postprocesses the buffer, "unwrapping" it if required, so that the oldest data is put at the beginning of the buffer, and the newest data is put at the end. Thus, the buffer wrap-around is invisible to the application.

5.8 Data Frames and Clusters

The PCI-20092S-1 Software Drivers handle the transfer of data by means of structures called **data frames**. Each data frame consists of two bytes, which contain the reading from one channel (one sample). One frame (one sample) is transferred at each tick of the Start Convert signal (for example, the PCI-20091W-1's onboard rate generator). Data from the various input channels in the data frames are grouped into **clusters**. Each cluster is associated with one pass through the channel list that you specified in the **CNF.DMA (Configure DMA)** instruction. The number of frames per cluster varies from one to eight, depending on how many channels you choose to read.

The PCI-20092S-1 Software takes care of organizing the **frame map** in accordance with the information you provide. The frame map is used to program the PCI-20091W-1, and to decode data frames. It is a list of the I/O channels which are to be involved in the DMA input of data. Data frames reside in the buffer in the host computer, and contain the actual data that is transferred to or from the I/O channels.

The DMA process transfers data into memory buffers allocated and controlled by the PCI-20092S-1 Software Drivers. The DMA process does not directly access your application program memory. The Software Drivers provide an instruction, **BUF.DEC (Decode Buffer)** that you may use to retrieve data from the buffer to your program memory. This instruction automatically formats the data and returns circular-buffer data in chronological order.

5.8.1 AUTO-SEQUENCED CHANNELS

A cluster consists of one to eight data frames; each successive data frame in a cluster will contain data from the next analog channel in the frame map. One complete pass through the list of analog channels constitutes a cluster. For example, if you have configured the PCI-20091W-1 for auto-sequencing, and you wish to sample channels 5, 6, and 7, each cluster will contain three data frames. In this case, the first data frame would contain the first analog channel (channel 5), the second would contain the second analog channel (6), and the third would contain the third analog channel (7).

5.8.2 DATA FRAMES AND DMA MODES 3 and 4

When a buffer is used for DMA Mode 3 or 4, the PCI-20092S-1 Software Drivers adjust the length of the circular buffer so that it contains an exact integral number of clusters. This ensures that when the buffer is overwritten, or "wraps," the first data frame in the buffer always contains the first analog channel, the second data frame contains the second analog channel, and so forth. This ensures that the data can be unambiguously associated with the correct channel sequence. In DMA Mode 3 (stop on trigger with delay), the delay is specified in *clusters*. This ensures that an integral number of clusters is always transferred.

5.9 I/O Types

Two I/O types are used to specify the DMA channel list. The first channel in the list always has I/O type **AI.T**, analog input. If the list contains more than one channel, subsequent channels have I/O type **AIX.T**. This is a special I/O type, not used by the **READ.CH (Read Channel)** instruction.

One pass through the complete list of AI.T and AIX.T channels results in a cluster of data frames, each data frame containing a different analog channel. That is, the first data frame in a cluster contains data from the AI.T channel; each successive data frame in the cluster contains data from the next associated AIX.T channel in the

frame map. The software ensures that the DMA buffer contains only complete clusters of data frames, and the **BUF.DEC (Decode Buffer)** instruction returns data in complete clusters.

5.10 DMA INSTRUCTIONS

The following sections briefly describe the instructions required for DMA. Although all these instructions are not required for all applications, the instructions required for a particular application should be executed in the order indicated.

5.10.1 DMA INITIALIZATION INSTRUCTIONS

CNF.DMA (Configure DMA). Call this instruction to initialize pacer source, DMA mode, trigger source, and frame map.

5.10.2 CONTROL INITIALIZATION INSTRUCTIONS

CNF.RG (Configure Rate Generator). Configure rate generator (if used) to start conversion prior to running DMA.

5.10.3 DMA EXECUTION INSTRUCTIONS

DMA.RUN (Run DMA). Start the DMA process with this instruction. In Mode 3, specify the trigger delay, in clusters, that you want between the trigger event and the end of the DMA process.

WRITE.CH (Write Channel). If you are using the onboard rate generator channel to start analog conversions, call this instruction to enable the rate generator **after** the call to **DMA.RUN.**

DMA.STAT (DMA Status). Use this instruction to check whether or not the DMA process has completed.

DMA.STOP (Stop DMA). This instruction may be used to stop the DMA process prior to its automatic termination.

5.10.4 BUFFER CONTROL INSTRUCTIONS

BUF.DEC (Decode Buffer). Call this instruction one or more times to read input data from the buffer. Specify the number of clusters to input in each call. You can call this instruction during DMA operation to return the most recently read cluster. This instruction returns data clusters from an input buffer. The buffer must contain data acquired by a DMA mode process. For example, first acquire data with a DMA Mode 3 process. When the process is complete, read back the data using **BUF.DEC.** To monitor data as it is taken, call **BUF.DEC** to return the single most recent cluster in the buffer.

CHAPTER 6: SOFTWARE TECHNICAL REFERENCE

6.1 Chapter Overview

This chapter covers in-depth technical information for your software drivers. Section 6.2 gives a high-level discussion of creating application programs in Microsoft QuickBASIC. Section 6.3 covers hardware initialization. Sections 6.4 and 6.5 provide a reference guide to the individual calls you will need to use. Section 6.6 lists and describes the error codes you may get.

6.2 Creating Application Programs

This section should get you started writing QuickBASIC application programs for your PCI-20091W-1 hardware. Before you begin to work on your own programs, you may wish to try out some of the Sample Programs on the PCI-20092S-1 diskette to familiarize yourself with the system.

When you start QuickBASIC, you must load the default User Library QB.QLB. This library contains the ABSOLUTE utility which calls PCI-20092S-1 instructions. To load the user library, type

```
QB /L
```

zat the DOS prompt. (Some versions of QuickBASIC require that you first create the User Library. Refer to your QuickBASIC documentation for details on Quick Libraries.)

You must use the header provided as PCIHEADQ.BAS at the beginning of your program. The easiest way to incorporate this file into your QuickBASIC program is to use the $INCLUDE metacommand

```
'$INCLUDE: 'PCIHEADQ.BAS'
```

in your program, as we have done in the Sample Programs listed in appendix A.

PCIHEADQ.BAS sets up constants, including the instruction names, that you may need in your program, and verifies that the correct version of PCI_92S.EXE was previously loaded in memory. This file also defines the interrupt vector that was used to load PCI_92S.EXE. If you specified an interrupt vector other than the default 60 (Hex) when you loaded PCI_92S.EXE, you must edit PCIHEADQ.BAS to specify that same vector.

Your program will control the PCI-20091W-1 hardware and read input data through QuickBASIC CALLS statement. You must be sure that all variables used in CALLS

statements are integers. The file PCIHEADQ.BAS declares all variables starting with letters other than **F** to be integers. (**F** is reserved for floating-point variables.)

Note that the header statement

```
DEF SEG = SEGMT
```

defines the segment used by the CALLS. statements. DEF SEG is also used in conjunction with PEEK and POKE statements. If your application program uses PEEKs and POKEs to access QuickBASIC's data space, or if you have CALLS or CALL statements to subroutines from some other software package, you must use the appropriate DEF SEG statements throughout your program to ensure that the correct segment is set for each CALLS, CALL, PEEK, and POKE statement.

The first calls you make in your program should ordinarily be

```
CALLS ABSOLUTE (SYSINIT)
CALLS ABSOLUTE (SEGMT, INIT)
```

You should call **INIT** for every PCI-20091W-1 you have installed in your system, setting the variable **SEGMT** to each segment address in turn that you have established with the address switches on each PCI-20091W-1. You can check for hardware configuration errors and other problems with the following:

```
ERROR.CODE = 0
CALLS ABSOLUTE (ERROR.CODE, ERR.SYS)
IF ERROR.CODE <> 0 THEN PRINT "Error: "; ERROR.CODE
```

Note that the variable ERROR.CODE is set to zero before the first call to **ERR.SYS**. A general-purpose error-checking subroutine is incorporated into the Sample Programs.

Now you are ready to add calls to read and write your system's input and output channels and perform various control functions. Until you are familiar with the PCI-20092S-1 instructions, you should add call statements a few at a time, with frequent calls to **ERR.SYS**. For example, to read analog input channel number CHN and place the returned value in VALUE, you could add:

```
CALLS ABSOLUTE (AI.T, CHN, VALUE, READ.CH)
CALLS ABSOLUTE (ERROR.CODE, ERR.SYS)
IF ERROR.CODE <> 0 THEN PRINT "Error: "; ERROR.CODE
```

You might also include program lines to display the data value returned in VALUE.

Save your program, and then run it and check for errors after each change. QuickBASIC provides no protection against the possibility of an instruction being called with the wrong number of arguments, or non-integer arguments. There is also nothing built into QuickBASIC to prevent you from attempting to call functions when PCI_92S.EXE has not been loaded, although the header contains routines to

check for a valid PCI_92S.EXE. To avoid losing your work if you inadvertently call an instruction incorrectly, you should save your program after every change involving calls.

6.3 Hardware Initialization

This section describes the system's hardware initialization sequence. This is important because the initialization sequence determines which channel number is used to reference each of the hardware channels.

6.3.1 INITIALIZATION SEQUENCE

The initialization is simple and straightforward. Each time your program executes the **INIT (Initialize)** instruction, the system first checks that a PCI-20091W-1 exists at the segment address given, and that an PCI-20091W-1 at that address has not previously been initialized. Then the system initializes the following channels:

8 Analog I/O Channels
1 Rate Generator Channel

All channel-number sequences begin with 0 as the first channel, 1 as the second channel, and so forth.

6.3.2 EXAMPLES OF HARDWARE INITIALIZATION

Example 1: One PCI-20091W-1

Consider a system with one PCI-20091W-1 installed.
The following channels will be configured:

Example 1 Channel Configuration

Channel Numbers	Channel Types
0 - 7	Analog Inputs
0	Rate Generator

Example 2: Two PCI-20091W-1 Boards

Consider a system with two PCI-20091W-1 Boards installed.
The following channels will be configured:

Example 2 Channel Configuration

Channel Numbers	Channel Types	Description
0 - 7	Analog Inputs	PCI-20091W-1 #1
8 - 15	Analog Inputs	PCI-20091W-1 #2
0	Rate Generator	PCI-20091W-1 #1
1	Rate Generator	PCI-20091W-1 #2

Example 3: PCI-20091W-1 with a PCI-20041C-3A Carrier

If you have a PCI-20091W-1 Board in the same system as you have a PCI-20041C-3A Carrier, the channels might be assigned as follows (assuming that you **INIT** the Carrier first, then the Board).

Example 3 Channel Configuration

Channel Numbers	Channel Types	Description
0	Rate Generator	PCI-20041C-3A
0 - 31	Digital I/O	PCI-20041C-3A
0 - 7	Analog Inputs	PCI-20091W-1
1	Rate Generator	PCI-20091W-1

6.4 Software Calls

This section provides a complete reference for the use of the PCI-20092S-1 Software Drivers system. The instruction summary below contains a quick-reference listing of the entire set of instructions available, accompanied by a brief explanatory description. Note that each instruction is listed by its actual call name, then by a more explanatory name and definition. Section 6.5, the **Instruction Specifications**, provides a detailed description of each instruction.

6.4.1 UTILITY INSTRUCTIONS

ERR.SYS (System Error)

Returns the latest error code and clears the error register.

SYNTAX CALLS ABSOLUTE (ERROR.CODE, **ERR.SYS**)

INIT (Initialize)

Initializes the system hardware.

SYNTAX CALLS ABSOLUTE (SEGMT, **INIT**)

SYSINIT (System Initialize)

Initializes the system software.

SYNTAX CALLS ABSOLUTE (**SYSINIT**)

6.4.2 CONFIGURATION INSTRUCTIONS

CNF.RG (Configure Rate Generator)

Configures the rate generator.

SYNTAX CALLS ABSOLUTE (CHN, COUNT1, COUNT2, MODE, **CNF.RG**)

6.4.3 READ INSTRUCTIONS

READ.CH (Read Channel)

Reads a value from a channel of any appropriate type.

SYNTAX CALLS ABSOLUTE (IO.TYPE, CHN, STOR, **READ.CH**)

6.4.4 WRITE INSTRUCTIONS

WRITE.CH (Write Channel)

Writes a value to a channel of any appropriate type.

SYNTAX CALLS ABSOLUTE (IO.TYPE, CHN, VALU, **WRITE.CH**)

6.4.5 BUFFER UTILITIES

BUF.DEC (Decode Buffer)

Decodes and returns the specified number of data units to the user's array.

SYNTAX CALLS ABSOLUTE (COUNT, DARRAY (1), XCOUNT, **BUF.DEC**)

6.4.6 HIGH SPEED ANALOG ACQUISITION INSTRUCTIONS

CNF.HS (Configure High Speed)

Configures system for high speed analog acquisition.

SYNTAX CALLS ABSOLUTE (PAC.TYPE, PAC.CHN, MODE,
TR.TYPE, TR.CHN, AI.CHNS(1), **CNF.HS**)

HS.RUN (Run High Speed)

Executes a high speed analog acquisition operation.

SYNTAX CALLS ABSOLUTE (PASS, DARRAY (1), SEGMNT, **HS.RUN**)

6.4.7 DMA DATA ACQUISITION AND CONTROL INSTRUCTIONS

CNF.DMA (Configure DMA)

Sets up a general DMA channel list.

SYNTAX CALLS ABSOLUTE (PAC.TYPE, PAC.CHN, MODE, TR.TYPE, TR.CHN, TYPES (1), CHNS (1), CNT, **CNF.DMA**)

DMA.RUN (Run DMA)

Executes a DMA-controlled data acquisition or output process.

SYNTAX CALLS ABSOLUTE (CLUSTERS, DELAY, **DMA.RUN**)

DMA.STAT (DMA Status)

Returns the status of a DMA run.

SYNTAX CALLS ABSOLUTE (COUNT, STATUS, **DMA.STAT**)

DMA.STOP (Stop DMA)

Stops the DMA operation.

SYNTAX CALLS ABSOLUTE (COUNT, **DMA.STOP**)

6.5 Instruction Specifications

This section contains a detailed listing of the instructions available in the PCI-20092S-1 Software Driver system. The listing is presented in alphabetical order by actual call, for ease of reference.

The specific syntax of each instruction is given. The semantic description of the arguments is included. The arguments are numbered in the order in which they appear in the given format. Some instructions, notably the **READ.CH, ERR.SYS, BUF.DEC, CNF.DMA, DMA.STAT,** and **DMA.STOP** instructions, pass a return value to the calling program. This value appears as the final argument in the argument list.

Table 6.1 lists the I/O types and what their names are.

TABLE 6.1 I/O Types

Definition	Name
Analog Input	AI.T
Analog input (auto sequencing)	AIX.T
Rate Generator/Pacer Channel	RG.T

The call to each instruction appears as it would be used in your program. Constants and variables used in PCI-20092S-1 CALLS statements **must** be **integers.** That means they either must be of the form "variable%", or a DEFINT command must exist in the program, or they must be declared as integers in a DIM statement.

BUF.DEC (Decode Buffer) — Buffer Instruction

PURPOSE

Decodes and returns the specified number of clusters from the buffer to the user's array, starting at the current location pointer. The location pointer is updated to the end of the data decoded.

SYNTAX

CALLS ABSOLUTE (COUNT, DARRAY (1), XCOUNT, BUF.DEC)

ARGUMENTS

1. (COUNT) Number of clusters to return.
2. (DARRAY (1)) Array for returned data.
3. (XCOUNT) Number of data units actually transferred. Be sure to predefine this argument by setting it equal to 0 prior to this call.

REMARKS

This instruction decodes COUNT clusters into the user's array DARRAY, starting at the current location in the buffer.

12-bit analog data is returned as an integer between 0 and 4095, in complementary binary or complementary offset binary format, right-justified, and zero-filled. If DMA is in progress and at least one cluster has been acquired, the most recent cluster will be returned. If no complete clusters have been read, XCOUNT returns 0 and no data is returned.

CNF.DMA (Configure DMA) — Configuration Instruction

PURPOSE

Prepare for DMA data acquisition. Set up general DMA channel list. Supports DMA Modes 2 and 3.

SYNTAX

CALLS ABSOLUTE (PAC.TYPE, PAC.CHN, MODE, TR.TYPE, TR.CHN, TYPES (1), CHNS (1), CNT, CNF.DMA)

ARGUMENTS

1. (PAC.TYPE) The I/O type of the pacing signal for the DMA process. The pacer type must always be AI.T.

2. (PAC.CHN) The channel number of the first channel on the PCI- 20091W-1 Board. This will be 0 unless your system contains more than one PCI-20091W-1.

3. (MODE) DMA Mode desired. Valid modes are 2, 3, or 4.

4. (TR.TYPE) The I/O type of the trigger source. The user may specify XSYNC, to use the external interrupt line on the PCI-20091W-1, or 0, for no trigger. Note only DMA Mode 3 requires a trigger. A trigger event occurs when the External Interrupt is taken low.

5. (TR.CHN) Trigger source channel number. Must be 0.

6. (TYPES) I/O type list corresponding to the channel list of Argument 7. See the discussion below for legal type list format.

7. (CHNS) List of I/O channel numbers, with the final balue being -1 to indicate the end of the list. See the discussion below for legal channel list format.

8. (CNT) Number of bytes per cluster, or 0 if errors were encountered. Be sure to predefine this argument by setting it equal to 0 prior to this call.

REMARKS

The Software Drivers use the information in the arrays TYPES and CHNS to set up the frame map and determine the number of frames per cluster. The number of *bytes* per cluster is returned as the final instruction argument.

The channel and type lists passed to **CNF.DMA** must follow certain conventions because of the way the PCI-20091W-1 hardware operates.

If the PCI-20091W-1 is configured with automatic channel advance disabled, the channel list may consist of any single channel of type AI.T.

If the PCI-20091W-1 is configured with automatic channel advance enabled, the channel list may consist of a channel sequence, in numerical order, beginning with N (where N <= 7) and ending with

7. The first channel, N, has type AI.T, and subsequent channels have type AIX.T.

The following examples illustrate each case.

SINGLE ANALOG INPUT CHANNEL

Channel 3 on the PCI-20091W-1 is read. The PCI-20091W-1 must be configured with automatic channel advance disabled.

Type	Channel Number
AI.T	3

SEVERAL ANALOG INPUT CHANNELS

Channels 4, 5, 6, and 7 are read. The PCI-20091W-1 must be configured with automatic channel advance enabled.

Type	Channel Number
AI.T	4
AIX.T	5
AIX.T	6
AIX.T	7

The channels on the following list are **NOT LEGAL**, because the channels are not consecutive. If **CNF.DMA** encounters a channel list like this, it will return an error.

ILLEGAL CHANNELS

Type	Channel Number
AI.T	2
AIX.T	5
AIX.T	6
AIX.T	7

CNF.HS (Configure High Speed) — Configuration Instruction

PURPOSE

Configures the system for high speed analog acquisition.

SYNTAX

CALLS ABSOLUTE (PAC.TYPE, PAC.CHN, MODE, TR.TYPE, TR.CHN, AI.CHNS(1), CNF.HS)

ARGUMENTS

1. (PAC.TYPE) The I/O type of the pacing signal used as the time base for high speed acquisition. This may be the standard I/O type of RG.T, or the external start-convert signal may be used by specifying PAC.TYPE of 0. The pacer must be independently configured; it will be armed or enabled as necessary by **HS.RUN (Run High Speed)**.

2. (PAC.CHN) Channel number of the pacing signal.

3. (MODE) High speed acquisition mode; must be 4.

4. (TR.TYPE) Must be 0.

5. (TR.CHN) Must be 0.

6. (AI.CHNS) List of analog input channel numbers (contained in an array), with the final array value being -1 to indicate the end of the list. This instruction only supports analog channels on a single PCI-20091W-1 Board. See the discussion below for legal channel list format.

REMARKS

High Speed Mode 4 uses hardware-controlled timing to maximize data throughput to the user's array. The A/D converter on the PCI-20091W-1 gets its start-conversion signal from the onboard rate generator or the External Start Convert. During data acquisition, the software polls the End Of Convert signal. When this signal is detected, the current channel is read and the data is stored. The software then resumes polling for the end-of-convert signal.

The Software Drivers use the information in the array AI.CHNS to set up the frame map and determine the number of frames per cluster.

The channel list passed to **CNF.HS** must follow certain conventions because of the way the PCI-20091W-1 hardware operates.

If the PCI-20091W-1 is configured with automatic channel advance disabled, the channel list may consist of any single analog channel.

If the PCI-20091W-1 is configured with automatic channel advance enabled, the channel list may consist of a channel sequence, in numerical order, beginning with N (where N <= 7) and ending with 7.

Refer to the examples of acceptable and unacceptable channel lists given under the **CNF.DMA (Configure DMA)** instruction.

CNF.RG (Configure Rate Generator) — Configuration Instruction

PURPOSE Configure a Rate Generator channel with two 16-bit initial counts, and an Intel 8254 Mode code.

SYNTAX **CALLS ABSOLUTE (CHN, COUNT1, COUNT2, MODE, CNF.RG)**

ARGUMENTS

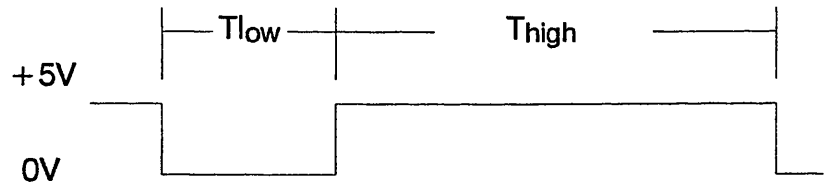
1. (CHN) Rate Generator channel to configure.

2. (COUNT1) Count data word, **LSW**, for low-order section of Rate Generator. A count of 1 is illegal in Modes 2, 3, 8, and 9.

3. (COUNT2) Count data word, **MSW**, for high-order section of Rate Generator. A count of 1 is illegal in Modes 2, 3, 8, and 9.

4. (MODE) Intel 8254 Mode code: 0 through 5 to select 16-bit Modes, 6 through 11 to select corresponding 8-bit Modes. Only Modes 2, 3, 8, and 9 are true rate-generator Modes.

REMARKS This instruction configures a specified rate-generator channel, writing the control words and initial counts to the hardware. Two internally connected counters are set by the two count arguments (**LSW** and **MSW**). The rate-generator waveform produced by these two counters operating in Intel mode 2 or 8 is shown in figure 8. Modes 3 and 9 produces a square-wave output. A rate generator channel must be configured using this instruction before it is enabled with the **WRITE.CH (Write Channel)** instruction.]



Rate Generator Waveform in modes 2 (CNF.RG)

Intel Mode 2:

T_{low} = **LSW** counts * $1/(8 * 10^6)$
T_{high} = **LSW** counts * **MSW** counts * $1/(8 * 10^6)$

Intel Mode 3:

$T_{low} = T_{high}$ = **LSW** counts * **MSW** counts * $1/(8 * 10^6)$

If a rate generator channel is used for high-speed data acquisition or DMA data acquisition, the data acquisition sampling rate in cycles per

second, **F**, is determined by the rate generator counts **LSW** and **MSW**. The sampling period in seconds, **T**, is given by

$$\mathbf{T} = 1 / \mathbf{F}$$

$$= (\mathbf{LSW} * \mathbf{MSW}) / (8 * 10^6)$$

The *minimum* values for **LSW** and **MSW**, corresponding to the *maximum* sampling rate, are determined by the hardware configuration and the data acquisition mode used.

DMA.RUN (Run DMA) — DMA Control Instruction

PURPOSE

Execute a DMA process set up by the **CNF.DMA (Configure DMA)** instruction.

SYNTAX

CALLS ABSOLUTE (CLUSTERS, DELAY, DMA.RUN)

ARGUMENTS

1. (CLUSTERS) This argument is used in DMA Mode 2 to specify the number of clusters of data to acquire. In DMA Modes 3 and 4, the number of clusters acquired is the maximum that will fit into the DMA buffer; CLUSTERS must be set to 0. (Note that in Modes 3 and 4 much more data may be read as new data overwrites old data.)

2. (DELAY) Specifies the number of clusters for the DMA process to transfer after the trigger event (DMA mode 3). Since the delay counter counts frames (pacer clock ticks), the count with which the delay counter is programmed will be determined by multiplying DELAY by the number of samples per cluster. DELAY should be set to 0 in DMA Modes 2 and 4.

REMARKS

None.

DMA.STAT (DMA Status) — DMA Control Instruction

PURPOSE Return the status of a DMA run.

SYNTAX CALLS ABSOLUTE (COUNT, STATUS, DMA.STAT)

ARGUMENTS

1. (COUNT) Number of clusters available for reading. Zero will be returned if the status return value, below, is non-zero. Be sure to predefine this argument by setting it equal to 0 prior to this call.

2. (STATUS) A zero will be returned if there is no DMA process running. A -1 will be returned if there was an error (other than overrun) associated with a DMA process. (The system error register will indicate an overrun condition, if one exists.) Other status codes, described below, indicate that a DMA process is running. Be sure to predefine this argument by setting it equal to 0 prior to this call.

Table 6.2 summarizes the status codes returned when this instruction is called during various phases of a DMA process.

TABLE 6.2 DMA Status Return Values

DMA Mode		Run DMA Command	Trigger Event	Delay Count	Terminal Count	Stop DMA Command
2	0	** 2 **	** 2 **	** 2 **	0	0
3	0	** 3 **	** 3 **	0	0	0
4	0	** 2 **	** 2 **	** 2 **	** 2 **	0

Notes: The vertical bars (|) indicate the time at which the indicated function or event occurred.

The asterisks (**) indicate phases during which data is being transferred via a DMA process.

REMARKS This instruction is useful for monitoring the status of a DMA operation. It DMA Modes 2 and 2, it may be used to determine whether the DMA process has completed. In all DMA modes, it checks for DMA overruns, and sets the error code if an overrun has occurred. The **COUNT** value is set only after a DMA process is complete. While the DMA process is active, **COUNT** is returned as 0.

In DMA Modes 2 and 3, in which the DMA process is stopped automatically, you *must* call an instruction that will cause the Software

Drivers to check the status of the DMA process after it halts. Such instructions include **DMA.STAT, DMA.STOP,** and **BUF.DEC.**

DMA.STOP (Stop DMA) — DMA Control Instruction

PURPOSE

Stop the DMA operation. This instruction is required in DMA Mode 4; in other DMA modes, this instruction will also stop the DMA process, but an error code will be set if acquisition was in progress. If an overrun condition exists, the error code for the overrun will supersede errors due to an acquisition in progress.

SYNTAX

CALLS ABSOLUTE (COUNT, DMA.STOP)

ARGUMENTS

1. (COUNT) Number of clusters available to be read. If errors other than an overrun were encountered, a count of 0 will be returned. Be sure to predefine this argument by setting it equal to 0 prior to this call.

REMARKS

If **DMA.STOP** is called in DMA Mode 2, an error code will be set if the DMA process has not been terminated by terminal count. In DMA Mode 3, an error code will be set if the DMA process has not been terminated by the trigger condition. Calling **DMA.STOP** in DMA Mode 4 will not set a system error code; this is the normal method of terminating a DMA process in DMA Mode 4. This instruction should be called prior to exiting your application program to ensure that all DMA activity is terminated.

ERR.SYS (System Error) — Utility Instruction

PURPOSE Return the latest error code, and clear the error register.

SYNTAX CALLS ABSOLUTE (ERROR.CODE, ERR.SYS)

ARGUMENTS 1. (ERROR.CODE) Error code value. Be sure to predefine this argument by setting it equal to 0 prior to this call. See sections 6.7 through 6.13 for an explanation of the error codes. There is no indication provided if previous error conditions have gone unchecked. The value 0 indicates no error.

REMARKS This instruction returns the contents of the error register as the return value. The contents of the error register reflect the last error that occurred. Note that this does not imply that the error is necessarily associated with the last instruction called, because the contents of the error register are changed only under three conditions: calling the **SYSINIT (System Initialize)** instruction, which sets the error register to 0; calling an instruction that causes an error code to be put in the error register; or calling **ERR.SYS**, which returns the error and then sets the error register to 0. If you wish to ensure that the error code returned is associated with a particular instruction, call **ERR.SYS** before calling that instruction (to set the error register to 0), and then again after calling that instruction (to return the error code, if any). It is good practice, when debugging your program, to call **ERR.SYS** after each PCI-20092S-1 call. In this way, your program will immediately be informed of any instruction errors, and the codes in the error register will always be up to date.

HS.RUN (Run High Speed) — Data Acquisition Instruction

PURPOSE

Execute a high speed analog acquisition operation.

SYNTAX

CALLS ABSOLUTE (PASS, DARRAY (1), SEGMNT, HS.RUN)

ARGUMENTS

1. (PASS) Indicates how many times the high speed acquisition routine is to go through the list of channels set up by the **CNF.HS (Configure High Speed)** instruction. The pass count parameter of this instruction may not exceed &HFFFF (65535 decimal), or 32767 if decimal integers are used.

2. (DARRAY) Array in which the acquired data is stored. Must be dimensioned to a size that will hold at least the number of channels in the list multiplied by the number of passes specified.

3. (SEGMNT) Must be 0.

REMARKS

Overrun conditions are not detected in High Speed Mode 4.

Note: HS.RUN disables all interrupts, including the computer timer. If you call **HS.RUN**, your computer's timer will not be correct.

INIT (Initialize) — Utility Instruction

PURPOSE

Initialize the PCI-20091W-1, setting up channel assignments for all I/O channels contained on the PCI-20091W-1, and performing hardware initialization where appropriate.

SYNTAX

CALLS ABSOLUTE (SEGMT, INIT)

ARGUMENTS

1. (SEGMT) Segment address (high-order 16 bits) in memory at which the PCI-20091W-1 Board is located, as determined by the Board's switch settings. See section 2.2.

REMARKS

This instruction initializes the system to support an installed PCI-20091W-1. All rate generator channels are disabled by the **INIT** call.

The PCI-20091W-1 must have previously been set up with its address switches to have its segment address specified by the argument. When multiple PCI-20091W-1 Boards are installed in your computer, **INIT** each of them, in *decreasing* order of segment addresses. Refer to section 6.3 for a detailed discussion of the initialization procedures for the PCI-20091W-1.

READ.CH (Read Channel) — Read Instruction

PURPOSE

Read the specified analog channel and return the value read.

SYNTAX

CALLS ABSOLUTE (IO.TYPE, CHN, STOR, READ.CH)

ARGUMENTS

1. (IO.TYPE) I/O type of the desired channel. Must be AI.T.

2. (CHN) Analog input channel number to read.

3. (STOR) Data read. Be sure to predefine this argument by setting it equal to 0 prior to this call. A return value of -1 may indicate an error condition.

REMARKS

This instruction reads the analog input channel specified by the CHN argument. An analog conversion is performed by software command. On completion, the data from the conversion is read and the input value is transferred to the return-value variable STOR.

The data returned by READ.CH may be converted to volts by means of the appropriate formula for PCI-20091W-1 analog inputs:

Volts = range bottom + ((4095-data) * **lsb**)

where **lsb** = (range top - range bottom)/4096.

ERRORS

Any error encountered during a read of an analog input will cause the returned data to be set to -1.

SYSINIT (System Initialize) — Utility Instruction

PURPOSE Initialize the Software Drivers system memory area. This instruction must be called prior to any other call.

SYNTAX CALLS ABSOLUTE (SYSINIT)

ARGUMENTS None.

REMARKS This instruction completely initializes the Software Drivers system, including setting up all of the internal variables, and initializing the system data area. **SYSINIT** is normally called only once during execution of a program.

WRITE.CH (Write Channel) — Write Instruction

PURPOSE Enable or disable a PCI-20091W-1 rate generator channel.

SYNTAX **CALLS ABSOLUTE (IO.TYPE, CHN, VALU, WRITE.CH)**

ARGUMENTS

1. (IO.TYPE) Integer specifying the I/O type of the channel to be written. Must be RG.T.
2. (CHN) Integer specifying the channel number to be written to.
3. (VALU) Data value to be written.

REMARKS This instruction writes to the output device specified by arguments 1 and 2. The third argument is written to the output. To enable a rate generator channel, write any non-zero value to that channel. To disable a channel, write a zero to it.

6.6 Software Error Codes

The PCI-20092S-1 Software Drivers are quite failure resistant. When a failure does occur, either because of internal error or because of inappropriate user requests, the system does not lock up. Instead, the system simply returns to the user's QuickBASIC program, and leaves the error-code register of the Software Drivers set with a number corresponding to the error that occurred. This error code can be read using the **ERR.SYS (System Error)** instruction.

The only exception is the case in which a Software Drivers instruction is called using an incorrect number of arguments. In this case, the return to QuickBASIC will be incorrect, and there is a possibility that the computer will lock up, requiring a system reset.

The error codes represent three types of errors. **Internal errors** are caused only by software inconsistencies. Any such errors should be reported to the factory immediately. **Hardware incompatibility errors** are primarily due to unrecognized PCI-20091W-1 IDs, and could result from using hardware which postdates the version of the Software Drivers that is being used. **User errors** result from improper calls by the user.

Sections 6.7.1 through 6.7.6 give the error-code values returned by the **ERR.SYS** instruction, and a brief description of each.

6.6.1 0 - 99. System Error Codes

Error Code	Description
0	No error condition since the last call to the system error code routine.
1	Internal error.
2	Bad function call. A call has been made to offset 0 in the Software Drivers segment. This usually occurs if a function name is incorrectly typed in a QuickBASIC program.
3	Invalid I/O type. The I/O type specified in a read channel or write channel call is not supported by this software. The only types supported by this software are AI.T and RG.T.
4	Invalid input type. Input from the specified I/O type is not supported by this software.
5	Invalid output type. Output to the specified I/O type is not supported by this software.
6	Invalid channel number. The specified channel number does not exist for the I/O type specified.

7	Duplicate PCI-20091W-1 initialization. Two calls to initialize the same Board (**INIT**) have been made. Call the system initialization function (**SYSINIT**) at the beginning of the program to clear the system memory.
8	Invalid PCI-20091W-1 ID. An invalid PCI-20091W-1 ID has been found at the segment address specified by the **INIT** call. Check your address swithc for proper settings.
9	Invalid minor PCI-20091W-1 ID. The minor PCI-20091W-1 ID code is not supported by this software. (This error code may appear if a PCI-20041C-3A Carrier is used with an unsupported Module.)
10	Internal error.
12	Invalid input channel. Input functions for this channel are not supported.
13	Invalid output channel. Output functions for this channel are not supported.
14	Invalid configuration. A configure function is not supported for this channel/type.
15	Re-entry error. An illegal re-entry to the Software Driver routines has been attempted. This can be caused by calling the Software Drivers routines from interrupt routines.
16	Test failure. A Board initialization test failed.

6.6.2 100 - 199. Analog Input Error Codes

101	Analog-to-digital (A/D) converter time-out. No end-of-conversion signal was received from an A/D converter.

6.6.3 800 - 899. Rate Generator Error Codes

801	Invalid rate generator mode. The rate generator mode specified is not supported for the specified channel.
802	Rate generator not configured. A rate generator must be configured prior to a write operation.
803	Illegal count value. The specified count value is not legal for the rate generator mode being configured. Eight-bit rate generator modes (modes 6 through 11) must be configured with count values less than 256. In modes 2, 3, 8, and 9, a count value of 1 is illegal.

6.6.4 15,000 - 15,799. High Speed Acquisition Error Codes

15100	Invalid trigger. High speed acquisition trigger operation is not supported for the specified channel.

15101 Illegal trigger address. The trigger channel must be on the same Board, or one at a higher segment address, as the analog input channels in the channel list. This can occur in a multi-Board system.

15102 Illegal trigger. The specified trigger type is not legal in High Speed Mode 4.

15200 Invalid time base. High speed acquisition timing operation is not supported for the specified channel.

15201 Illegal time base address. The pacer channel must be on the same PCI-20091W-1, or one at a higher segment address, as the analog input channels in the channel list.

15203 Illegal time base. The specified time base typ is not legal in this High Speed mode.

15204 llegal rate generator mode. The rate generator is configured in a mode which is illegal for a pacer.

15300 No High Speed Mode 1 function. Data acquisition in High Speed Mode 1 is not supported by this software.

15301 List empty. The first element of the analog channel list was -1, the list terminator.

15302 Invalid high speed mode. The only high speed mode supported is Mode 4.

15303 Analog configuration error. All analog input channels in the channel list must be reside on the same Board.

15304 High speed configuration error. A successful high speed configuration was not performed prior to attempting a high speed acquisition.

15305 Invalid pass count. A pass count of 0 was specified, or a pass count high enough to produce more than 64K of data was specified for a buffer not aligned on an even 16-byte paragraph.

15307 Internal error.

15309 Pass count too large. The pass count specified would cause the data acquisition buffer size to be exceeded.

15400 No High Speed Mode 2 function. Data acquisition in High Speed Mode 2 is not supported by this software.

15500 No High Speed Mode 3 function. Data acquisition in High Speed Mode 3 is not supported by this software.

15601 Illegal channel sequence.

15701 Illegal I/O type. The specified pacer or trigger I/O type is not supported by the PCI-20091W-1 Board.

6.6.5 15,800 - 16,499. DMA Error Codes

15800 No DMA function. The PCI-20091W-1 Board has not been jumper configured for DMA operation.

15801 No DMA buffer. The PCI-20092S-1 software was loaded without a DMA buffer. The software package should be removed and reloaded, specifying a buffer of the desired size.

15804 Configuration error. All specified I/O channels are not on the same Board.

15805 Invalid mode. An incorrect DMA acquisition mode was specified.

15806 DMA configuration error. A successful DMA configuration was not performed prior to attempting a DMA acquisition, read, status check, or stop.

15807 Invalid pass count. A pass count of 0 was specified, or the pass count specified would exceed the size of the data acquisition buffer.

15809 DMA overrun. The DMA timing signal supplied is too fast. Slow down the acquisition.

15810 Trigger active. The DMA Trigger delay was still active when DMA acquisition was stopped. In Mode 3, the trigger location in the buffer will not be correct.

15811 DMA active. DMA acquisition was active when DMA acquisition was stopped, in modes other than stop on command. In Mode 2, fewer data were transferred than requested. Alternatively, DMA acquisition was active when an attempt was made to start acquisition.

15814 No DMA data. The **BUFDEC** was called prior to calling the DMA run function.

15817 Delay count too large. The delay count, converted from cluster count to frame count, exceeded 65536.

15818 Internal error.

15823 Internal error.

16000 No DMA list function. DMA acquisition is not supported for a specified channel.

16001 DMA list empty. The first entry in the DMA channel array was -1.

16002 Illegal list configuration. The I/O type sequence in the DMA list is incorrect.

16005 Cluster size error. Two analog input channel sequences were specified.

16100 Invalid trigger. DMA acquisition trigger operation is not supported for the specified channel.

16101 No trigger. DMA Mode 3 requires that a trigger be specified.

16102 Invalid trigger code. An invalid I/O code was used to specify the DMA trigger. The only I/O codes supported for the PCI-20091W-1 is XSYNC.

16200 Invalid time base. DMA acquisition timing operation is not supported for the specified channel.

16201 Invalid time base code. No time base was specified, or an invalid I/O code was used.

6.6.6 20,100 - 20,199. Buffer Error Codes

20100 Internal Error.

20105 Data acquisition process running. An attempt was made to modify or read buffer parameters for a buffer actively in use for data acquisition. Halt all data acquisition processes before modifying buffers.

20107 Internal error.

20108 Internal error.

20109 Invalid cluster size. An attempt was made to initialize a buffer with a cluster size greater than the buffer size.

20110 Internal error.

20115 No memory has been allocated for buffers. Reload the Software Drivers specifying **/B** on the command line to allocate the desired amount of buffer memory.

CHAPTER 7: HARDWARE TECHNICAL REFERENCE

7.1 Introduction and Chapter Overview

This chapter contains technical details of your hardware, including register information, that you may need for creating application programs if you choose not to use the included software or the PCI-20026S and PCI-20027S Software Drivers. Section 7.2 tells how to test and calibrate your Board. Sections 7.3 and 7.4 cover the details of addresses, segments, and registers (offsets). Section 7.5 gives the specific details on how to program your hardware using the registers. Section 7.6 gives a summary table of the jumper assignments. Be sure to read the descriptions in chapter 2 before using this table. Section 7.7 gives the hardware specifications.

7.2 Testing and Calibration

Your Board comes from the factory configured and calibrated for a range of ± 10 volts. If you change this configuration, or if you suspect inaccuracies, *check* the calibration on your Board. If the results of this test are out of specification and unacceptable for your application, use the following procedure to recalibrate.

EQUIPMENT NEEDED

1. Voltage calibrator accurate to 0.5 mV over a range of ± 10 V.
2. 4 1/2 digit digital multimeter.
3. Small insulated screwdriver for adjusting potentiometers. **Warning!** If you use an uninsulated screwdriver and accidentally touch other components, you may damage your Board.
4. A test program to sample the input continuously, and display the results.
5. IBM PC, IBM XT, IBM AT, or equivalent.
6. The PCI-20091W-1 High Speed Analog Input Board.
7. PCI-20012A Analog Cable or equivalent.
8. The PCI-20010T Analog Termination Panel or equivalent.

PROCEDURE

1. Turn off the power to the computer.

WARNING
Lethal voltages exist inside computers. Always ensure that power is removed before opening the case.

CAUTION
Failing to turn off the power when inserting or removing boards will damage the boards, and possibly the computer as well.

Reading errors of more than 5 to 10 counts suggest a test procedure difficulty. Check your setup carefully before making large adjustments.

2. Remove the cover from the computer, following the manufacturer's instructions.

3. If you haven't already done so, install your Board (see chapter 2).

4. Connect any additional cables into the Termination Panel(s).

5. Turn on the power to the computer. Allow at least 10 minutes warm-up time before continuing.

6. Connect the calibrator to Channel 0, and set its output following the guidelines given in table 7.1.

TABLE 7.1 PCI-20019M-1A Calibration Set Points

Module Input Range	Calibrator Output
±5 V	-4.9988 V
0 to 10 V	+1.22 mV
±10 V	-9.9976 V
0 to +5 V	+0.610 mV
±2.5 V	-2.4994 V

7. Start the sampling program. Adjust Pot R4 until the reading displayed on the computer is alternating between 4094 and 4095.

8. Set the output of the calibrator using the data in table 7.2.

TABLE 7.2 PCI-20019M-1A Calibration Set Points

Module Input Range	Calibrator Output
±5 V	+4.9963 V
0 to 10 V	+9.9963 V
±10 V	+9.9927 V
0 to +5 V	+4.9982 V
±2.5 V	+2.4982 V

9. Adjust Pot R3 until the reading alternates between 0 and 1.
10. Replace the cover on your computer.
11. End of test and calibration procedure.

7.3 Doing It Yourself: Hardware Programming

We highly recommend our PCI-20026S, PCI-20027S, and PCI-20092S Software Drivers. These Software Drivers are efficient and provide most functions that you might wish to implement with your board, and they can save you a great deal of time. However, if you wish to write your own software, the following sections provide important information.

The PCI-20091W-1 Board has certain functions that are programmed by reads and writes to various registers (offsets). While a discussion of the individual offsets is given in section 7.4, this section gives an overview of the operation of these functions, and of the offsets needed to properly program these functions.

7.3.1 ANALOG-TO-DIGITAL (A/D) CONVERTER

The A/D converter provides an A/D Status signal which goes low when a conversion is completed. It is set high when the most significant byte (MSB) of data is read. The A/D status can be read in Offset 01. When the conversion is completed, the data must be read before another conversion is started or the data will be lost. A read of Offset 101 (Hex) returns the least significant 8 bits of data. A read of Offset 102 (Hex) returns the most significant 4 bits of the 12-bit conversion in a right-justified format.

7.3.2 RATE GENERATOR

The PCI-20091W-1 provides a rate generator that outputs a variable duty cycle (mode 2) or square wave (mode 3) TTL level signal. The output period is set by the following formula:

$$\text{Period} = (N1 * N2) / (8 * 10^6)$$

where **N1** is the rate generator low-order 16 bits, and **N2** is the rate generator high-order 16 bits. **N1** and **N2** can be programmed with values from 2 to 65536 to produce a wide range of output frequencies. In the variable duty cycle mode (mode 2) the low time of the output, **T1**, and the high time of the output, **T2**, are determined by

$$T1 = N1 / (8 * 10^6)$$

$$T2 = N1 * (N2 - 1)\ (8 * 10^6)$$

N1 is set by writing to Offset 40 (Hex), and **N2** is set by writing to Offset 41 (Hex). The operating modes and functions of the rate generator are set by writing the

proper control word to Offset 43 (Hex). The status of the Rate Generator output signal can be monitored by reading Offset 01.

7.3.3 DMA OPERATION

The Board contains DMA control circuitry allowing it to communicate with the PC's DMA controller to sample analog input data under complete hardware control. The Board's DMA mode is programmed to one of three modes by writing to Offset 02. DMA data transfers are started and stopped by reads to Offset 02 and Offset 03, respectively. Status of a DMA operation, including the occurance of DMA overruns, can be monitored by reading Offset 01.

7.3.4 BOARD IDENTIFICATION

Several offsets and jumpers are concerned with different identification codes. Offset 000 holds a value for an ID code which identifies the Board as either a PCI-20041C Carrier with one Module installed, or as a PCI-20091W-1. Offset 004 (Hex) uniquely identifies the Board as a PCI-20091W-1. Offset 100 (Hex) provides an ID code that "identifies" a PCI-20019M Module as being present. See the discussion under each of these offsets for more information.

7.4 Register Offsets

The computer controls the functions of the PCI-20091W-1 by accessing registers at memory mapped offsets within the Board's 1 Kbyte block of memory. The programming language you use must have the capability of absolute memory reference.

7.4.1 I/O REGISTERS

The registers may be read only, write only, or read and write registers. In addition, the same register may be used for two different functions, depending on whether it is read or written. If the register is read only, you will not be able to change its contents directly. If the register is write only, or if its read function is unrelated to its write function, you will not be able to read back data that was previously written to the register; you must save the data in program memory in order to reference it later. In some cases, just reading or writing a register may trigger an event, such as an analog conversion, regardless of the data involved. Be careful not to inadvertently read or write such registers, particularly if you use a debugger during program development.

7.4.2 MEMORY ADDRESSES

The 8086 microprocessor family addresses memory using a 16-bit SEGMENT register in conjunction with a 16-bit OFFSET to specify a 20-bit ADDRESS, as follows:

ADDRESS = SEGMENT * 16 + OFFSET

Most programming languages reflect this addressing scheme in their provisions for absolute memory reference. Consult your programming manual to find out how to read and write absolute memory locations. It does not matter how you divide the address specification between SEGMENT and OFFSET, but for simplicity we will assume that the SEGMENT is chosen so that the base address of the Carrier is at OFFSET 0. When OFFSET is 0, the 20-bit ADDRESS will correspond exactly to the setting of the base address switches on the Board. Information on how to set the address switches can be found in chapter 2.

7.4.3 REGISTER OFFSETS

The register offsets from the Board's base address are given in table 7.3. All register addresses are expressed in hexadecimal. This table also tells whether the registers are read, write, or read/write, and summarizes each register's functions. The remainder of the section describes each of registers individually.

TABLE 7.3 Register Offsets

Read/Write Offset (hex)	(R/W)	Function
000	R	ID Code 1
	W	Interrupt/Status Mode
001	R	System Status
	W	Interrupt/Status Clear
002	R	DMA Start
	W	DMA Mode
003	R	DMA Stop
004	R	ID Code 2
005 - 03F		Reserved
040	R/W	Low Order Pacer Counter
041	R/W	High Order Pacer Counter
042	R/W	Delay Counter
043	W	Pacer/Delay Control Register
044 - 0FF		Reserved
100	R	ID Code 3
101	R	LSB analog data
102	R	MSB analog data
103	R/W	Start Conversion
104	W	Input Channel
105	W	Scanner Start Channel

7.4.4 REGISTER USE

A detailed description of the use of each register follows (offsets given in hex):

Offset 000: Board ID; Interrupt/Status Mode

READ: Board ID, bits 0 - 7

Reading this register returns one of the following ID codes, which can be interpreted as follows:

ID code	DMA enabled	Terminal Count interrupt	Extended
D7	YES	NO	NO
D3	NO	NO	NO
D1	YES	YES	NO
C4	YES	NO	YES
C5	YES	YES	YES

The first three ID codes are equivalent to a PCI-20041C-3A Carrier with a Module in slot 1. The extended IDs, C4 (Hex) and C5 (Hex) are incompatible with older versions of our software drivers (the PCI-20046S, PCI-20047S, and older versions of the PCI-20092S Software Drivers). The different IDs are selected by jumpers, as shown in table 7.4.

TABLE 7.4 DMA Control Jumper Settings

ID Code	Jumper IN	Jumpers OUT
DMA without terminal count	---	W16, W17, W18
Non-DMA mode	W16	W17, W18
DMA with terminal count	W17	W16, W18
Extended without terminal count	W16, W17	W18
Extended with terminal count[1]	W18	W16, W17

[1] Factory configuration

WRITE: Interrupt/Status Mode, bit 7

Writing an appropriate value to bit 7 of this register sets the interrupt mode. A value of 80 (Hex) (bit 7 set) sets latched interrupt mode. A value of 0 (Hex) (bit 7 clear) sets pass-through interrupt mode. Bits 0 through 6 are not used.

bit 7: 0 = pass through
1 = latched

When the Interrupt Mode is set for latched operation, the Pacer, External Interrupt, and Terminal Count status bits (read from Offset 001, below) are latched. The

External Interrupt hardware interrupt signal is also latched. The A/D converter End Of Convert status bit is always latched.

Offset 001: System Status; Interrupt/Status Clear

READ: System Status, bits 0 - 5

Reading this register returns system status information, which can be interpreted using the information below. The DMA terminal count signal must be latched to be used properly during DMA input.

bit 5: AI EOC (A/D converter End of Conversion)
bit 4: Rate generator output status
bit 3: EXT INT interrupt status (External Interrupt)
bit 2: DMA terminal count interrupt status
bit 1: 1 = DMA overrun
bit 0: 1 = DMA in progress

WRITE: Interrupt/Status Clear, data ignored

Writing any data to this register clears an interrupt generated from the External Interrupt input, and also clears the System Status Register.

Offset 002: DMA Start; DMA Mode

READ: DMA Start, data undefined

Reading this register initiates a new DMA transfer sequence. (The data read has no meaning.)

WRITE: DMA Mode, bits 0 - 5

Writing to this offset sets the DMA Mode, and enables or disables the DMA Trigger Delay, the Rate Generator, and DMA operations, as shown below.

bit 5: 0 = init DMA
1 = enable DMA
bit 4: 0 = disable Delay
1 = enable Delay
bit 3: 0 = disable Rate Generator
1 = enable Rate Generator
bits 0 to 2: DMA mode number
DMA Mode 2: 010
DMA Mode 3: 011
DMA Mode 4: 100

The Rate Generator and the Delay counter should be enabled only after the counters have been properly configured, as described under Offset 43.

Following a complete DMA data acquisition, the Board's DMA circuitry must be reset before starting another acquisition. Write a 0 and then a 1 to bit 5 to initialize and re-enable DMA operation.

Offset 003: DMA Stop

READ: DMA Stop, data undefined

Reading this register will stop a DMA process in case of an overrun error. This is the normal termination of a Mode 4 DMA process only. (The data read has no meaning.)

WRITE: NOT USED

Offset 004: ID Code 2

READ: ID Code 2, bits 0 - 7

Reading this register returns 87 (Hex). This code can be used to uniquely identify this Board as being a PCI-20091W-1, rather than a PCI-20041C-3A Carrier/PCI-20019M-1A Module combination. See section 1.4 for an explanation of how the Board emulates this Carrier/Module combination.

WRITE: NOT USED

Offsets 005 - 03F: Reserved

Offset 040: Low Order Rate Generator Counter

Before any reads or writes are performed from this register, the Rate Generator control register must be configured (see Offset 043).

READ:

Reading this offset returns the current value of the low order 16 bits of the Rate Generator counter. The data must be read as two consecutive bytes from the register, reading the least significant byte (LSB) first.

The programmed mode and status of the counter can be read from this register using the read back command described under Offset 43.

WRITE:

Writing to this offset sets the initial value of the low order 16 bits of the Rate Generator counter. The data must be written as two consecutive bytes to the register, writing the least significant byte (LSB) first.

Offset 041: High Order Rate Generator Counter

Before any reads or writes are performed from this Offset, the Rate Generator control register must be configured (see Offset 043).

READ:

Reading this offset returns the high order 16 bits of the Rate Generator counter. The data must be read as two consecutive bytes to this offset, reading the least significant byte (LSB) first.

The programmed mode and status of the counter can be read from this register using the read back command described under Offset 43.

WRITE:

Writing to this offset sets the high order 16 bits of the Rate Generator counter. The data must be written as two consecutive bytes to this offset, writing the least significant byte (LSB) first.

Offset 042: Delay Counter

Before any reads or writes are performed from this register, the Delay counter control register must be configured (see Offset 043).

READ:

Reading this register returns the current value of the Delay counter. The data must be read as two consecutive bytes to the register, reading the least significant byte (LSB) first.

The programmed mode and status of the counter can be read from this register using the read back command described under Offset 43.

WRITE:

Writing to this register sets the Delay counter. The data must be written as two consecutive bytes to the register, writing the least significant byte (LSB) first.

Offset 043: Rate Generator and Delay Counter Control

READ: Cannot be read. The programmed counter mode can be read from the count registers using a read back command (described below).

WRITE: Rate Generator and Delay Counter Modes

Writing to this offset defines the mode of both the Rate Generator and Delay counters. The Rate Generator clock counters (Offsets 040 and 041) should be programmed in Intel Modes 2 or 3. See the discussion of the **CNFRG (Configure Rate Generator)** instruction in chapter 6 for details. The Delay counter, used for DMA, must be configured in Intel Mode 0. The values for the control word to perform these operations are given in table 7.5.

TABLE 7.5 Offset 043 Control Words

Bit	Name	Function
7	SC1	00 = Rate generator, LSB
6	SC0	01 = Rate generator, MSB
		10 = Delay
		11 = read back command
5	RW1	00 = counter latch command
4	RW0	01 = LSB only
		10 = MSB only
		11 = LSB then MSB
3	M2	000 = Intel Mode 0
2	M1	001 = Intel Mode 1
1	M0	010 = Intel Mode 2
		011 = Intel Mode 3
		100 = Intel Mode 4
		101 = Intel Mode 5
0	BCD	0 = Binary
		1 = BCD

To program the Rate Generator in Intel Mode 2 (2-byte LSB then MSB data transfer), write the codes given in table 7.6 to Offset 043.

TABLE 7.6 Offset 043 Rate Generator (RG) Programming Codes

If you wish to program these registers to the Intel Mode at right:		Program Offset 043 to this Hex code:	
Register	Offset	Intel Mode	Control Code
RG, LSW	040	2	34 Hex
RG, LSW	040	3	36 Hex
RG, MSW	041	2	74 Hex
RG, MSW	041	3	76 Hex

After the counters have been configured, the count registers may be initialized or read. To read the counter's data or status, the read back command should be issued to the control register. The format of the read back command is shown in table 7.7.

TABLE 7.7 Offset 043 Counter Data and Status Read Command Format

Bit	Name	Function
7	SC1	1 = read back command
6	SC0	1 = read back command
5	/COUNT	0 = latch count
4	/STATUS	0 = latch status
3	Delay	1 = select Delay counter
2	RG, MSB	1 = select MSB of RG
1	RG, LSB	1 = select LSB of RG
0	0	

To set the Delay counter for read back, the command word to be written to Offset 043 is 11001000 (C4 (Hex)). This latches both the count and the status of the Delay counter. The status is returned in the first read of the Delay counter (Offset 042). The next two reads from the Delay counter give the two count bytes. The format of the returned status is given in table 7.8.

TABLE 7.8 Offset 040 or 041 Returned Status Format

Bit	Name	Function
7	OUTPUT	0 = OUT pin low 1 = OUT pin high
6	NULL COUNT	1 = null count
5	RW1	| programmed
4	RW0	| Read/Write mode
3	M2	| programmed
2	M1	|
1	M0	| Intel Mode
0	BCD	| programmed BCD

Bits 0 - 5 are equal to the previously programmed mode of the counter. **Note:** The term "Intel Mode" refers to the Intel 8254 counter/timer modes. This mode is distinct from "DMA Mode."

Offsets 044 - 0FF: Reserved

Offset 100: ID Code 3

READ: ID Code 3, bits 0 - 7

Reading this register returns an ID code. This code indicates whether or not the Board is configured for automatic channel scanning. See section 2.5 for a discussion of configuring automatic channel scanning.

C4 Automatic channel scanning enabled
C5 Automatic channel scanning disabled

These are the same ID codes that are returned from a PCI-20019M-1A High Speed Analog Input Module. They must be used in order to maintain compatibility with older versions of the software drivers (PCI-20046S and PCI-20047S, and older versions of PCI-20092S Software Drivers).

WRITE: NOT USED

Offset 101: LSB Analog Data

READ: LSB Analog Input Data, bits 0 - 7

Reading this register returns analog conversion data that is right justified in the following format:

D7	D6	D5	D4	D3	D2	D1	D0
MSB							LSB

WRITE: NOT USED

Offset 102: MSB Analog Data

READ: MSB Analog Input Data, bits 0 - 3

Reading this offset returns data that is right justified in the following format:

XX	XX	XX	XX	D11	D10	D9	D8
MSB							LSB

Reading this offset also clears the AI EOC bit in the Interrupt/Status register. If the Board is configured for start of conversion on read of MSB, reading this register will start a new analog-to-digital (A/D) conversion. See section 2.4 for information on configuring this mode.

WRITE: NOT USED

Offset 103: Start Conversion

Reading from or writing to this register initiates a software analog input start convert.

READ: Start Conversion, data undefined

Reading this offset starts a conversion. (The data read has no meaning.)

WRITE: Start Conversion, data ignored

Writing to this offset starts a conversion. (The data written has no meaning.)

Offset 104: Input Channel

READ: NOT USED

WRITE: Input Channel, bits 0 - 3

Writing to this offset sets the analog input channel to be converted on the next hardware or software start convert command. If automatic channel sequencing is enabled, the first channel in the sequence input channel must be set by writing to this register as well as by programming Offset 105.

bit 3: 0 = disable MUX
1 = enable MUX

bits
0 -2: channel number of analog input channel

Offset 105: Scanner Start Channel

READ: NOT USED

WRITE: Scanner Start Channel, bits 0 - 2

If the PCI-20091W-1 Board is configured for Hardware Scan mode (see section 2.5), the starting channel must be written to this register. In addition, Offset 104 must be programmed with the first channel to be read.

bits 0 through 2: channel number of analog input channel

7.5 Hardware Programming Procedures

7.5.1 MEMORY ADDRESSES

The following procedures are given as representative examples for programming this Board. These procedures are not intended to be an exhaustive list of programming possibilities.

In the programming procedures that follow, the address of each register is expressed as **(BA+n)**, where **n** is the register offset, and **BA** is the Board's block address. Single-byte references are identified as **BYTE** (*address*), and two-byte references are identified as **WORD** (*address*). For example, a word at local register offset 102 (Hex) on a Board would be identified as **WORD (BA+102H)**. A byte at local register offset 040 (Hex) on a Board would be identified as **BYTE (BA+040H)**.

7.5.2 ANALOG INPUT PROCEDURE

The following procedure performs a conversion on an analog input channel and reads the converted data.

Step 1. Make sure the Interrupt/Status register is reset by reading BYTE (BA+102H). Ignore the data read.

Step 2. Set the multiplexers by writing the appropriate byte to BYTE (BA+104H).

Step 3. Wait for the input amplifier and multiplexer to settle. You will probably not need to do this unless you are programming in assembly language on a high-performance computer. You can ensure an adequate settling delay by entering a count-down loop with the number of loop cycles adjusted suitably.

Step 4. Initiate a conversion by writing any data to BYTE (BA+103H).

Step 5. Wait for the conversion to complete. To test whether the conversion is complete, read the conversion status in the appropriate bit of BYTE (BA+001H). Depending on the speed of your CPU, this step may not be necessary.

Step 6. Read the converted data from WORD (BA+101H).

Repeated measurements of the same channel can be performed by repeating Steps 4 through 6.

7.5.3 DATA ACQUISITION WITH HARDWARE TIMING

This procedure uses a hardware pacing source (rate generator or external start convert) for pacing the acquisition. It is assumed that the board is jumpered for automatic channel scanning. The Conversion Inhibit input may be used to gate the acquisitions.

This procedure may be used to sample a single channel (7), from one to eight channels (N - 7), or all eight channels (0 - 7).

Step 1. Make sure the Interrupt/Status register is reset by reading BYTE (BA+102H). Ignore the data read.

Step 2. Set the initial channel by writing the appropriate byte to BYTE (BA+104H).

Step 3. Program the automatic channel scanner by writing the same channel number you used in Step 2 to BYTE (BA+105H).

Step 4. Wait for a hardware-initiated conversion to complete. To test whether the conversion is complete, read the conversion status in the appropriate bit of BYTE (BA+001H).

Step 5. Read the converted data from WORD (BA+101H).

Repeat Steps 4 and 5 until data acquisition is complete.

7.5.4 DELAY COUNTER INITIALIZATION PROCEDURE

This procedure initializes the Delay counter.

Step 1. Write the control byte B0H to the control register, BYTE (BA+043H). This programs the delay counter for Intel Mode 0, binary code, and 16-bit read-write format. The count register is reset, counting is disabled, and the output signal is set low.

Step 2. Write the initial count to the count register, BYTE (BA+042H). Write first the low byte and then the high byte of the count. Both bytes are written to the same register: this is not a WORD operation.

7.5.5 DELAY COUNTER ENABLE OR DISABLE PROCEDURE

This procedure enables or disables the Delay counter.

Step 1. Set up the enable byte. Since this register controls the Delay counter, the Rate Generator clock, and the DMA operation, you must be sure to set *all* bits correctly. The enable byte should have each bit that corresponds to the Delay counter or rate generator clock section set to a 1 if the section is to be enabled, and to a 0 if the section is to be disabled.

Step 2. Write the enable byte to the DMA mode register, BYTE (BA+002H). This register cannot be read, so you must store the enable byte in program memory.

7.5.6 DELAY COUNTER READ PROCEDURE

This procedure latches and reads the current counter status and count. (Other read-back options are available, and the counter can also be read without latching the count.)

Step 1. Set up the read-back control byte. To read the counter status and count, this byte should be C8H. (The count and status can optionally be read separately.)

Step 2. Write the control byte to the appropriate control register, BYTE (BA+043H). This latches the status and the count, which will be the next three bytes read from the count register.

Step 3. Read the status byte from the count register, BYTE (BA+042H).

Step 4. Read the count bytes from the count register. Read first the low byte and then the high byte of the count. Both bytes are read from the same register: this is not a WORD operation.

If no pulses have been received at the counter clock input since the initial count was programmed, bit 6 of the status byte will be 1, and the count read will be indeterminate. After one pulse is received, the count read will be the initial count that was programmed. Thereafter, the initial count will be decremented for each pulse received. The count is decremented by one or two, depending on the operating mode.

7.5.7 RATE GENERATOR INITIALIZATION PROCEDURE

This procedure initializes the Rate Generator.

Step 1. Write the first control byte (36H for mode 3; 34H for mode 2) to the control register, BYTE (BA+043H). This resets the count register of the Rate Generator section and disables counting. The output signal is set high.

Step 2. Write the second control byte (76H for mode 3; 74H for mode 2) to the control register, BYTE (BA+043H).

Step 3. Write the first count to the first count register, BYTE (BA+040H). Write first the low byte and then the high byte of the count. Both bytes are written to the same register: this is not a WORD operation.

Step 4. Write the second count to the second count register, BYTE (BA+041H). Write first the low byte and then the high byte of the count. Both bytes are written to the same register: this is not a WORD operation.

7.5.8 RATE GENERATOR ENABLE OR DISABLE PROCEDURE

This procedure enables or disables the Rate Generator.

Step 1. Set up the enable byte. Since this register controls the Delay counter, the Rate Generator, and DMA operation, you must be sure to set *all* bits correctly. The enable byte should have each bit that corresponds to the Delay counter or Rate Generator section set to a 1 if the section is to be enabled, and to a 0 if the section is to be disabled.

Step 2. Write the enable byte to the enable register, BYTE (BA+002H). This register cannot be read, so you must store the enable byte in program memory.

7.5.9 DMA PROCEDURES

In the DMA procedures below it is assumed that the DMA pacing source is the on-board rate generator. If the External Start Convert input is used as the pacing source, omit the rate generator programming. In this case, to keep the input channels from becoming rotated, the start convert signal should only be applied after DMA programming is complete and the DMA circuitry is enabled. This can be accomplished by using the Convert Inhibit input to gate the start convert signal.

7.5.10 DMA MODE 2 PROCEDURE

This procedure describes programming the Board for DMA Mode 2 operation. If the External Start Convert input is used as the pacing source, omit the rate generator programming.

Step 1. First, make sure that the source of A/D start convert signals is disabled. The on-board rate generator can be disabled by clearing bit 3 of BYTE(BA+002).

Step 2. Set the on-board rate generator frequency as described in section 7.5.6, RATE GENERATOR INITIALIZATION PROCEDURE.

Step 3. Reset the PCI-20091W's DMA controller by writing 00H to BYTE (BA+002).

Step 4. Configure the PC's DMA controller for no auto-initialize operation, and set the base address, byte count registers, and terminal count.

Step 5. Program DMA Mode 2 and enable DMA operation on the PCI-20091W by writing 22H to BYTE(BA+002).

Step 6. Set up the analog input channel sequence as described in section 7.5.2, DATA ACQUISITION WITH HARDWARE TIMING.

Step 7. Enable the IBM PC's DMA.

Step 8. Start the DMA process by reading BYTE(BA+002).

Step 9. Enable the rate generator, as described in section 7.5.7, RATE GENERATOR ENABLE OR DISABLE PROCEDURE. The Board's DMA operation must be started (Step 8) before starting the rate generator, or the analog input channels may be rotated.

Step 10. The DMA process will stop after the specified number of bytes have been transferred. You can determine when the process reaches terminal count by monitoring bit 2 of BYTE(BA+001).

7.5.11 DMA MODE 3 PROCEDURE

This procedure describes DMA Mode 3 operation. If the External Start Convert input is used as the pacing source, omit the rate generator programming.

Step 1. First, make sure that the source of A/D start convert signals is disabled. The on-board rate generator can be disabled by clearing bit 3 of BYTE(BA+002).

Step 2. Set the on-board rate generator frequency as described in section 7.5.6, RATE GENERATOR INITIALIZATION PROCEDURE.

Step 3. Reset the PCI-20091W's DMA controller by writing 00H to BYTE (BA+002).

Step 4. Set up the delay counter as described in section 7.5.3 DELAY COUNTER INITIALIZATION PROCEDURE and section 7.5.4, DELAY COUNTER ENABLE OR DISABLE PROCEDURE.

Step 5. Configure the PC's DMA controller for auto-initialize operation, and set the base address, byte count registers, and terminal count. When the DMA controller reaches the programmed terminal count, it will reset its address to the base address and continue.

Step 6. Program DMA Mode 3 and enable DMA operation and the Delay Counter on the PCI-20091W by writing 33H to BYTE(BA+002).

Step 7. Set up the analog input channel sequence as described in section 7.5.2, DATA ACQUISITION WITH HARDWARE TIMING.

Step 8. Enable the IBM PC's DMA.

Step 9. Enable the DMA process by reading BYTE(BA+002).

Step 10. Enable the rate generator, as described in section 7.5.7, RATE GENERATOR ENABLE OR DISABLE PROCEDURE. The Board's DMA operation must be started (Step 9) before starting the rate generator, or the analog input channels may be rotated.

Step 10. The DMA process will stop after the specified number of bytes have been transferred. You can determine when the process reaches terminal count by monitoring bit 2 of BYTE(BA+001).

Step 11. The DMA process will stop after the specified delay from the DMA Trigger event. You can determine when the process reaches terminal count by monitoring bit 7 of the Delay Count Status register, which can be read as described in section 7.5.5 DELAY COUNTER READ PROCEDURE.

7.5.12 DMA MODE 4 PROCEDURE

This procedure describes DMA Mode 4 operation. If the External Start Convert input is used as the pacing source, omit the rate generator programming.

Step 1. First, make sure that the source of A/D start convert signals is disabled. The on-board rate generator can be disabled by clearing bit 3 of BYTE(BA+002).

Step 2. Set the on-board rate generator frequency as described in section 7.5.6, RATE GENERATOR INITIALIZATION PROCEDURE.

Step 3. Reset the PCI-20091W's DMA controller by writing 00H to BYTE (BA+002).

Step 4. Configure the PC's DMA controller for auto-initialize operation, and set the base address, byte count registers, and terminal count. When the DMA controller reaches the programmed terminal count, it will reset its address to the base address and continue.

Step 5. Program DMA Mode 4 and enable DMA operation on the PCI-20091W by writing 24H to BYTE(BA+002).

Step 6. Set up the analog input channel sequence as described in section 7.5.2, DATA ACQUISITION WITH HARDWARE TIMING.

Step 7. Enable the IBM PC's DMA.

Step 8. Start the DMA process by reading BYTE(BA+002).

Step 9. Enable the rate generator, as described in section 7.5.7, RATE GENERATOR ENABLE OR DISABLE PROCEDURE. The Board's DMA operation must be started (Step 8) before starting the rate generator, or the analog input channels may be rotated.

Step 10. Stop the DMA process by reading BYTE(BA+003).

7.6 Jumper Definitions Summary

Table 7.9 summarizes the jumper functions discussed in chapter 2. Once you have grown familiar with the PCI-20091W-1, you may wish to refer to this table for changing your Board's configuration. Factory settings are indicated in boldface type.

TABLE 7.9: Jumper Definitions Summary

Jumper	Function	
W4	IN	Enables DMA Terminal Count interrupt
	OUT	**Disables DMA Terminal Count interrupt**
W5	IN	Enables External Interrupt
	OUT	**Disables External Interrupt**
W6	**IN**	\
W7	**OUT**	| These jumpers select the A/D
W8	**OUT**	- Range and Polarity (see
W9	**IN**	| section 2.3).
W10	**OUT**	/
W12	**IN**	**Enables rate generator start convert**
	OUT	Disables rate generator start convert
W13	IN	Enables start convert on read of MSB
	OUT	**Disables start convert on read of MSB**
W14	**IN**	**Enables channel scanner**
	OUT	Disables channel scanner
W15	IN	Disables channel scanner
	OUT	**Enables channel scanner**
W16	**OUT**	\ These jumpers select capabilities
W17	**OUT**	- and Board ID codes (see section
W18	**OUT**	/ (see section 2.7).
W19	IN	Disables channel scanner
	OUT	**Enables channel scanner**
W20	IN	Enables external start convert
	OUT	**Disables external start convert**

Table 7.9 cont'd

JUMPER	DEFINITION	
W21	IN	Enables DMA channel 2
	OUT	**Disables DMA channel 2**
W22	IN	Enables PC interrupt level 3
	OUT	**Disables PC interrupt level 3**
W23	IN	Enables PC interrupt level 4
	OUT	**Disables PC interrupt level 4**
W24	**IN**	**Enables PC interrupt level 5**
	OUT	**Disables PC interrupt level 5**
W25	IN	Enables PC interrupt level 6
	OUT	**Disables PC interrupt level 6**
W26	IN	Enables PC interrupt level 7
	OUT	**Disables PC interrupt level 7**
W27	IN	Enables DMA channel 1
	OUT	**Disables DMA channel 1**
W28	IN	Enables DMA channel 1
	OUT	**Disables DMA channel 1**
W29	IN	Enables DMA channel 3
	OUT	**Disables DMA channel 3**
W30	IN	Enables DMA channel 3
	OUT	**Disables DMA channel 3**
W31	**IN**	**Enables DMA channel 2**
	OUT	**Disables DMA channel 2**
W32	IN	Enables PC interrupt level 2
	OUT	**Disables PC interrupt level 2**

7.7 Hardware Specifications

ANALOG INPUTS	
Offset Voltage	Trimmable to 0 ± 1/2 LSB
Offset Drift	
Unipolar	± 3 ppm/°C
Bipolar	± 15 ppm/°C
Impedance	1 Mohm, single ended
Voltage Range	
Linear	± 10 V
Without Damage	20 V above supplies
Bias Current	100 nA
Noise	± 1 LSB on the ± 2.5 volt range, shorted inputs
EXTERNAL CONTROL SIGNALS	
External Start Convert	TTL compatible
Active Edge	High to Low to start conversion
Conversion Inhibit	TTL compatible
Active Level	Low to inhibit
External Interrupt	TTL compatible
Active Level	Low to interrupt
DYNAMIC RESPONSE	
Mux Settling Time to 0.01%	3.5 μsec max
A/D Conversion Time	10 μsec max
Acquisition Time to 0.01%	1.5 μsec max
Aperture Jitter	0.3 nsec max
Total Conversion Time	11.25 μsec typ
Crosstalk	1/2 LSB

A/D CONVERTER

Input Range		
	Unipolar	0 to 5 V, 0 to 10 V
	Bipolar	± 2.5 V, ± 5 V, ± 10 V
Resolution		12 bits
Output Code		
	Unipolar	Complementary Binary
	Bipolar	Complementary Offset Binary
Linearity Error		± 1/2 LSB
Drift		± 3 ppm/°C
Accuracy		Trimmable to 0 ± 1/2 LSB
Drift		± 30 ppm/°C

A/D CODES

To calculate the value for the voltage being measured, substitute the A/D output code (INDATA) from table 12 into the appropriate formula listed in table 7.10

TABLE 7.10 A/D Conversion Formulas

Voltage Range	*Conversion Formula*
± 2.5V Range:	VOLTS = ((4095 - INDATA) * 5 / 4096) - 2.5
± 5V Range:	VOLTS = ((4095 - INDATA) * 10 / 4096) - 5
± 10V Range:	VOLTS = ((4095 - INDATA) * 20 / 4096) - 10
0 TO 5V Range:	VOLTS = (4095 - INDATA) * 5 / 4096
0 TO 10V Range:	VOLTS = (4095 - INDATA) * 10 / 4096

For example, if the conversion code is 2047, and the PCI-20019M-1A is jumpered for a bipolar range of ± 10 volts, then:

VOLTS = ((4095 - **2047**) * 20/4096 - 10

= 0

TABLE 7.11 A/D Codes

VOLTS Unipolar	*A/D Output Code (INDATA)*
Full Scale Minus One LSB	0
.	.
.	.
.	.
1/2 Full Scale	2047 (7FF Hex)
.	.
.	.
.	.
0	4095 (FFF Hex)
VOLTS Bipolar	*INDATA*
Full Scale Minus One LSB	0
.	.
.	.
.	.
0	2047 (7FF Hex)
.	.
.	.
.	.
Negative Full Scale	4095 (FFF Hex)
RATE GENERATOR	
Frequency Range	2 MHz - ~ 0.002 Hz
Frequency Accuracy	± 0.01% @25°C
Output Signal	TTL compatible
POWER REQUIREMENTS	
+5 V	1.7 A typ, 2.5 A max
PHYSICAL (L x H)	7.5" x 4.2"
OPERATING TEMPERATURE	0 - 70°C

Index

A

ABSOLUTE QuickBASIC utility 6.1
ABSOLUTE utility 3.5
Accuracy
  calibration 7.1
Address
  base 2.1
Address switches 2.1
  definitions 2.2
  factory configuration 2.1
  recommended settings 2.2
Addresses
  memory 7.4
AI.T I/O type 6.22
Analog connector 2.12
Analog data
  registers 7.12
Analog data acquisition
  high speed 4.1
Analog input
  channel selection 1.2
  convert inhibit 1.2
  pacing sources 1.2
  procedure with hardware timing 7.14
  programming procedure 7.13
  read channel 6.22
  selecting voltage ranges 2.6
  voltage ranges 1.1
Analog input channel
  register 7.13
Analog to digital converter
  hardware function 7.3
Application programs 6.1
  errors 6.2
  header files and 6.1
  sequence of calls 6.2
Automatic channel advance
  DMA 6.9
  High speed data acquisition 6.11
Automatic channel sequencing
  registers 7.13

B

Background operation
  DMA 5.4
Backup
  Software Drivers 3.3
Base address 2.1
  and register offsets 7.5
Batch file
  loading Software Drivers from 3.11 - 3.12
Batch file processing 3.13
Bit assignments
  analog input channel register 7.13
  delay counter control register 7.10
  DMA mode register 7.7
  Interrupt/Status mode register 7.6
  rate generator control register 7.10
  scanner start channel register 7.13
  system status register 7.7
Block diagram
  hardware 1.2
Board ID
  extended 7.6
  hardware function 7.4
  jumpers 7.6
  PCI-20019M-1A Module 7.11
  registers 7.6, 7.8
  unique 7.6
Bracket slot connector 2.12
BUFDEC instruction 5.6, 6.6, 6.8
Buffers
  and page boundary 5.5
  circular 5.5
  data 5.1
  data organization 5.5
  DMA modes 3 and 4 ..... 5.6
  DMA modes and 5.5
  for DMA 5.4
  linear 5.5
  maximum size 5.5
  reading 6.8
  wrap-around 5.5

C

Calibration 7.1
CALLS QuickBASIC command 6.1, 6.7
Channel advance
  automatic and manual 6.9, 6.11
Channel list
  DMA 6.9

high speed data acquisition 6.11
illegal for DMA 6.10
Channel numbers
analog input 2.12
assigned by software 6.3
numbering sequence 6.3
Channel programming
register.....7.13
Channel scanner start
register.....7.13
Channel scanning sequence
configuration 2.7
factory configuration.....2.7
jumpers.....2.7
register.....7.11
Channel selection
automatic sequencing.....1.2
high speed data acquisition 4.2
Channel sequencing
and clusters.....5.6
Chapter overview
chapter 1.....1.1
chapter 2.....2.1
chapter 3.....3.1
chapter 4.....4.1
chapter 5.....5.1
chapter 6.....6.1
chapter 7.....7.1
Circular buffers.....5.5
data frames and clusters.....5.6
Clock
rate generator.....2.10
Clusters.....5.5
circular buffers.....5.6
CNFDMA instruction 6.6, 6.9
CNFHS instruction.....6.11, 6.20
CNFRG instruction.....6.13
Command-line errors 3.9
COMMAND.COM DOS file.....3.8
COMP DOS command.....3.4
Compatibility with PCI-20000 system 1.3
Compatibility with Software Drivers....7.12
Computer memory map.....2.1
Configuration
automatic channel sequencing.....2.7
DMA control 2.9
hardware.....2.1
high speed acquisition.....4.1
interrupts.....2.7
start of conversion source.....2.6
Configuring DMA
software.....6.9
Connector
analog (P2).....2.12
digital (P3) 2.12
Constants
defined in PCIHEADQ.BAS 6.1
types 6.7

D

Data acquisition
procedure with hardware timing ...7.14
Data frames 5.5
buffers and 5.6
circular buffers 5.6
frame map and.....5.6
DEF SEG QuickBASIC command.....6.2
DEFINT QuickBASIC command.....6.7
Delay counter
configuring 2.10
enable or disable 7.15
initialization 7.14
programming procedure .....7.14 - 7.15
read procedure 7.15
registers.....7.9
status register.....7.11
Device drivers
software performance and.....3.10
Digital connector 2.12
DIM QuickBASIC command 6.7
Disable procedure
delay counter.....7.15
rate generator 7.16
DISKCOMP DOS command 3.3
DISKCOPY DOS command 3.3
DMA
buffers 5.4
channel channel configuration 2.9
channel list.....5.6
checking status.....6.16
continuous mode 5.3
control configuration 2.9
delay counter.....2.10
effect on foreground programs.....5.4
fixed-period mode 5.2
instructions.....5.7
jumpers 5.2
maximum sampling rates.....5.4
overrun error.....5.3 - 5.4
pacing signal 2.10
programming 5.7
sampling rate 5.3
sampling rate (formula).....6.14
sequence of operation.....5.1

setting delay counter....6.15
setting terminal count....6.15
software configuration....6.9
source event....5.2
speed....5.3
starting acquisition....6.15
stopping....6.18
system resource requirements....5.1
trigger event....5.2
triggered mode....5.2
DMA channel list
examples....6.9
illegal....6.10
DMA channel selection
factory configuration....2.10
DMA control
hardware function....7.4
DMA data
reading....6.8
DMA enable jumpers....2.9
DMA mode
register....7.7
DMA mode 2....5.2
checking termination....6.16
procedure....7.16
stopping before terminal count....6.18
DMA mode 3....5.2, 5.6
checking for trigger....6.16
procedure....7.17
stopping before trigger condition....6.18
DMA mode 4....5.3
procedure....7.18
stopping....6.18
DMA mode 4....5.6
DMA modes
buffers and....5.5
PCI-20047S Software Drivers....2.9
DMA programming procedure
external start convert pacing....7.16
rate generator pacing....7.16
DMA start
register....7.7
DMA stop
register....7.8
DMA terminal count
interrupt....2.8
DMA.RUN instruction....6.15
DMA.STAT instruction....6.6, 6.16
DMA.STOP instruction....6.6, 6.18
DOS
batch file processing....3.13
COMMAND.COM file....3.8
COMP command....3.4
DISKCOMP command....3.3
DISKCOPY command....3.3
EXIT command....3.8
MODE command....3.10
MORE command....3.3
PATH command....3.10
PRINT command....3.10
PROMPT command....3.10
SET command....3.10
user-defined interrupts....3.11
version required....3.2

E

Emulation of PCI-20041C Carrier....1.3
Enable procedure
delay counter....7.15
rate generator....7.16
Entry point offsets
mnemonic definitions....3.5
Environment
error message....3.10
ERR.SYS instruction....3.5, 6.6, 6.19, 6.25
Error codes....3.5, 6.25
list....6.25
Errors
command-line....3.9
during application program
execution....6.2, 6.19, 6.25
during channel reads....6.22
environment size....3.10
file not found....3.9
in arguments to CALLS....6.2, 6.25
subroutine to check....6.2
while loading command processor....3.9
while loading the Software Drivers....3.9
Examples
DMA channel list....6.10
hardware initialization....6.3
loading the software....3.12
EXIT DOS command....3.8
Expanded memory
interrupt vector....3.11
Extended ID....7.6
Extended mode....2.9
External start convert....1.2
DMA programming procedure....7.16
jumpers....2.7
start of conversion source....2.6

F

Factory configuration
address switch 2.1
channel scanning sequence 2.7
DMA channel selection 2.10
interrupt level 2.8
interrupt source 2.8
start conversion source 2.6
voltage ranges 2.6
Features
analog input 1.1
automatic channel selection 1.2
Convert Inhibit 1.2
DMA trigger 1.2
interrupts 1.2
rate generator 1.2
software 1.2
summary 1.1
Files
list 3.2
Foreground programs
and DMA 5.4
Forground programs
and DMA 5.4
Frame map 5.6
Frames 5.5
Frequency reference
rate generator 2.10
Functions
A/D converter 7.3
board ID 7.4
DMA operation 7.4
hardware 7.3
rate generator 7.3

H

Hard disk
subdirectory for Software Drivers 3.4
Hardware
block diagram 1.2
configuration and installation 2.1
initialization, examples of 6.3
installation 2.11
programming procedures 7.13
specifications 7.20
technical reference 7.1
testing and calibration 7.1
Hardware configuration
for high speed data acquisition 4.1
Hardware initialization
by software 6.3
examples 6.3
Hardware requirements 1.2
Hardware start convert
configuring 2.6
High speed channel list 6.11
High speed data acquisition
capabilities 4.1
channel selection. 4.2
computer lock-up 4.2
configuring in software 6.11
hardware configuration 4.1
instructions 4.2
Mode 4 ..... 4.3
overview 4.1
sampling rate (formula) 6.14
sequence of operation 4.4
testing sampling rates 4.3
time base 4.1
timing 4.3
timing signal 6.11
trigger signal 6.11
trigger source 4.2
High Speed Mode 4
overview 4.1
High Speed Modes 1 and 3 ..... 4.1
High-level language support 7.3
HS.RUN instruction 6.20

I

I/O type codes
mnemonic definitions 3.5
I/O types
DMA channel list 5.6
READ.CH 6.22
table 6.7
WRITE.CH 6.24
ID code
Board 7.6
configuring 2.9
ID codes
discussion 7.4
Illegal channels 6.10
INCLUDE QuickBASIC command 6.1
INIT instruction 3.5, 6.3, 6.21
Input channel
register 7.13
Installation
hardware 2.1
software 3.1

Installation procedure 2.1
Instruction offsets
  mnemonic definitions 3.5
Instructions 6.1
  BUF.DEC 5.7, 6.6, 6.8
  buffer utility 6.5
  CNF.DMA 5.7, 6.6, 6.9
  CNF.HS 4.2, 6.11
  CNF.RG 4.2, 5.7, 6.13
  configuration 6.5
  configure DMA 5.7, 6.9
  configure high speed 4.2, 6.11
  configure rate generator 4.2, 5.7, 6.13
  decode buffer 5.7, 6.8
  DMA (list) 5.7
  DMA status 5.7, 6.16
  DMA.RUN 5.7, 6.15
  DMA.STAT 5.7, 6.6, 6.16
  DMA.STOP 5.7, 6.6, 6.18
  ERR.SYS 3.5, 4.2, 6.6, 6.19
  high speed 6.5
  high speed 4.2
  HS.RUN 4.2, 6.20
  INIT 3.5, 6.3, 6.21
  initialize 6.21
  list 6.4
  read 6.5
  read channel 6.22
  READ.CH 3.5, 6.6, 6.22
  return value 6.6
  run DMA 5.7, 6.15
  run high speed 4.2, 6.20
  specifications 6.6
  stop DMA 5.7, 6.18
  summary 6.4
  SYSINIT 3.5, 6.23
  system error 4.2, 6.19
  system initialize 6.23
  utility 6.4
  write 6.5
  write channel 5.7
  write channel 6.24
  WRITE.CH 3.5, 5.7, 6.24
Interrupt
  terminal count enable 2.9
Interrupt level
  factory configuration 2.8
  jumpers 2.8
  selecting 2.8
Interrupt Request
  definition 2.7
Interrupt source
  factory configuration 2.8
  jumpers 2.8
  selecting 2.8
Interrupt vector
  setting in PCIHEADQ.BAS 3.11, 6.1
  setting with /V 3.11
  used by expanded memory 3.11
Interrupt/Status clear
  register 7.7
Interrupt/Status mode
  register 7.6
Interrupt/Status mode register 7.6
Interrupt/Status register 7.12
Interrupts
  capability 1.2
IRQ0*
  definition 2.7

J

Jumpers
  Board ID 7.6
  caution 2.1
  DMA channel 2.9
  DMA channel selection 2.10
  DMA configuration 5.2
  DMA control 2.9
  DMA enable 2.9
  DMA terminal count interrupt 2.8
  extended mode 2.9
  external start convert 2.7
  ID code 2.9
  interrupt level 2.8
  interrupt source 2.8
  polarity 2.6
  Rate Generator 2.7
  read MSB data 2.7
  start of conversion 2.7
  summary 7.19
  terminal count interrupt 2.9
  voltage range 2.6
  W11 through W13 ..... 2.7
  W14 and W15 ..... 2.7
  W16 through W18 ..... 2.9
  W19 ..... 2.7
  W20 ..... 2.7
  W21 ..... 2.10
  W22 through W26 ..... 2.8
  W32 ..... 2.8
  W4 and W5 ..... 2.8
  W6 through W10 ..... 2.6
Jumpers W27 through W31 ..... 2.10

L

Language requirements 3.2
Lethal voltages 7.2
License agreement 3.2
Linear buffers 5.5
Loading the software 3.8
  examples 3.12
Lock-up of computer
  high speed data acquisition 4.2

M

Manual channel advance
  DMA 6.9
  High speed data acquisition 6.11
Manual overview 1.1
Memory addresses 7.4
Memory map
  IBM PC 2.1
Memory resident programs 3.10
Mnemonic definitions 3.5
MODE DOS command 3.10
MORE DOS command 3.3

N

Naming conventions
  variables 3.2

O

Offset
  memory address 7.4
Offsets
  entry point 3.5
  hardware registers 7.4
Operation of Software Drivers 3.5
Overrun
  high speed data acquisition 6.20
Overrun error
  DMA 5.3

P

Pacing signal
  DMA 2.10
Pacing sources 1.2
Page boundary 5.5
PATH DOS command 3.10
PCI-20000 System
  compatibility 1.3
PCI-20026S Software Drivers 3.1
  compatibility with PCI-20091W-1 ..... 1.3
PCI-20026S Software Drives
  description 1.2
PCI-20027S
  high speed modes 4.1
PCI-20041C Carrier emulation 1.3
PCI-20046S Software Drivers
  compatibility with PCI-20091W-1 ..... 1.3
PCI-20047S Software Drivers
  DMA modes 2.9
PCI-20091W-1
  minimum requirements 1.2
  summary of features 1.1
  system compatibility 1.3
PCI-20092S
  requirements 3.2
PCI-20092S Software Drivers
  brief description 1.2
  description 3.1
  DMA modes 2.9
PCI_92S.EXE
  described 3.5
PCIHEADQ.BAS
  application programs and 6.1
  description 3.5, 6.1
PEEK QuickBASIC command 6.2
POKE QuickBASIC command 6.2
Polarity
  jumpers 2.6
  selecting 2.6
Posttrigger data 5.3
Pretrigger data 5.3
PRINT DOS command 3.10
Program license agreement 3.2
Programming 7.3
  DMA 5.7
Programming codes
  delay counter control register 7.10
  rate generator control register 7.10
Programming procedure
  delay counter initialization 7.14
Programming procedures 7.13
  analog input 7.13
  analog input settling delay 7.14
  delay counter enable or disable 7.15
  delay counter read 7.15
  DMA Mode 2 ..... 7.16

DMA Mode 3 ..... 7.17
DMA Mode 4 ..... 7.18
rate generator enable or disable 7.16
rate generator initialization 7.16
PROMPT DOS command 3.10

Q

QuickBASIC
ABSOLUTE utility 3.5, 6.1
INCLUDE command 6.1
user library 6.1
variables 6.1
QuickBASIC application programs 6.1
QuickBASIC commands
CALLS 6.7
CALLS and CALL 6.1
DEF SEG 6.2
DEFINT 6.7
DIM 6.7
PEEK 6.2
POKE 6.2
QuickBASIC variables 6.7

R

Ranges
selecting 2.6
Rate generator
configuring 6.13
DMA programming procedure 7.16
enable or disable 7.16
frequency determination 6.13
frequency reference 2.10
hardware function 7.3
initialization 7.16
jumpers 2.7
modes 6.13
output signal 1.2
programming procedure 7.16
programming procedures 7.16
registers ..... 7.8 - 7.9
start conversion source 1.2
start of conversion source 2.6
status register 7.11
time base for high speed data acquisition 4.1
waveform 6.13
Read back command
delay counter 7.11
rate generator 7.11
Read MSB
jumpers 2.7
start conversion source 1.2
start of conversion source 2.7
Read procedure
delay counter 7.15
READ.CH instruction 3.5, 6.6, 6.22
Reading channels
analog input 6.22
errors 6.22
Reading hardware registers 7.4
Register offsets 7.4
list 7.5
Registers
analog input channel 7.13
automatic channel sequencing 7.13
Board ID 7.6
channel scanning sequence 7.11
delay counter 7.9
delay counter control 7.9
delay counter status 7.11
description 7.6
DMA mode 7.7
DMA start 7.7
DMA stop 7.8
ID code 2 ..... 7.8
ID code 3 ..... 7.11
Interrupt/Status clear 7.7
Interrupt/status mode 7.6
LSB analog data 7.12
MSB analog data 7.12
offset list 7.5
rate generator control 7.9
Rate generator LSW 7.8
Rate generator MSW 7.8
rate generator status 7.11
reading and writing 7.4
scanner start channel 7.13
start of conversion 7.12
system status 7.7
Requirements
for PCI-20091W-1 ..... 1.2
for PCI-20092S 3.2
software language 3.2
Return value
instructions 6.6
RG.T I/O type 6.24

S

Sampling rate
DMA 5.3

formula 6.14
Sampling rates
DMA maximum 5.4
high speed data acquisition 4.3
test procedure 4.3
Scanner start channel
register 7.13
Segment
memory address 7.4
Sequence of operation
DMA 5.1
high speed data acquisition 4.4
Settling delay 7.14
Shell programs 3.10
Software
installation 3.1
Software command
start conversion source 1.2
Software drivers 7.3
channel numbers assigned 6.3
compatibility 7.12
DMA channel support 2.10
DOS commands and 3.10
errors while loading 3.9
general theory of operation 3.5
hardware initialization 6.3
instruction specifications 6.6
introduction 3.1
list of calls 6.4
list of files 3.2
loading 3.8
loading from batch file ..... 3.11 - 3.12
structure 3.5
technical information 6.1
unloading 3.8, 3.13
using in application programs 6.1
working copy 3.3
Software license agreement 3.2
Software performance
device drivers and 3.10
Software requirements 3.2
Software start convert 2.6
configuring 2.7
read MSB 2.7
register 7.12
Source event
DMA 5.2
Specifications
hardware 7.20
Speed
data acquisition 1.1
Start of conversion
choosing source 2.6
DMA 2.10
factory configuration 2.6
jumpers 2.7
register 7.12
software sources 2.7
sources (list) 1.2
Subdirectory for Software Drivers 3.4
Switches
base address 7.4
QuickBASIC 6.1
setting addresses 2.1
Switches (software)
/B 3.11
/G 3.11
/V 3.11
/X 3.12
allocate DMA buffer 3.11
bypassing confirmation 3.11
execute command line 3.12
go 3.11
interrupt vector 3.11
summary 3.12
SYSINIT instruction 3.5, 6.23
SYSINIT instruction 6.19
System resources
used by DMA 5.1
System status
register 7.7

T

Terminal count
interrupt 2.8
Testing
DMA sampling rates 5.4
hardware 7.1
Testing sampling rates
high speed data acquisition 4.3
Time base
high speed data acquisition 4.1
Timing
data acquisition 6.14
high speed data acquisition 4.3, 6.11
Trigger
high speed data acquisition 6.11
Trigger event
DMA 5.2
Trigger functions 1.2
Trigger source
high speed data acquisition 4.2
TSR programs 3.10

U

Unique Board ID
register....7.8
Unique ID....7.6
Unloading Software Drivers....3.8, 3.13

V

Variables
declaring in QuickBASIC....6.1, 6.7
naming conventions....3.2
types....6.7
Voltage calibration....7.1
Voltage range
jumpers....2.6
Voltage ranges
analog input....1.1
selecting....2.6
Voltages
lethal....2.11, 7.2

W

Windowing programs....3.10
Working copy
on two floppy drives....3.3
Software Drivers....3.3
with one floppy and one fixed drive.3.4
Write instructions....6.5
WRITE.CH instruction....3.5, 6.24
Writing hardware registers....7.4
Writing QuickBASIC application programs

Appendix A: Sample Programs

A.1 Appendix Overview

Sections A.1 through A.4 introduce the sample programs we have included with your Software Drivers in order to help you optimize your usage of your hardware and software. Section A.2 summarizes the sample programs and their associated instructions. Section A.3 describes use of these programs with Microsoft QuickBASIC. Section A.4 describes the graphics demonstration program.

A.2 SAMPLE PROGRAMS: INTRODUCTION

The Sample Programs on your Software Drivers diskette are described briefly below. After you have set up your PCI-20000 system and have loaded the PCI-20092S software, you might want to try out those programs that illustrate the hardware features you will use for your application. The Sample Programs are extensively commented, and contain many PRINT statements. You may find it instructive to study the Sample Programs before you begin to write your own application programs.

As you run the Sample Programs, you should supply signals to hardware inputs and monitor the outputs, as required by the programs.

A.3 Sample Programs: Instructions

The Sample Programs are intended to help you understand how to use the software in combination with your hardware. Each sample program illustrates the use of certain hardware features in combination with certain Software Drivers instructions. This section contains a list of the sample programs by number, with the software calls used by each program.

All sample programs use the following instructions:

ERR.SYS (System Error)

INIT (Initialize)

SYSINIT (System Initialize)

A.3.1 SAMPLE PROGRAM 1 DESCRIPTION: ANALOG INPUT

This program reads all eight analog input channels on the Board. The Board should be configured for +/- 10 volts. You should provide known voltages on at least some of the channels which you can verify against the values reported by the program.

Instructions used: **READ.CH (Read Channel).**

A.3.2 SAMPLE PROGRAM 2 DESCRIPTION: PULSE TRAIN

This program outputs a pulse train from the rate generator. You should monitor the rate generator output with a scope or frequency counter.

Instructions used: **CNF.RG (Configure Rate Generator)**
WRITE.CH (Write Channel).

A.3.3 SAMPLE PROGRAM 3 DESCRIPTION: HIGH SPEED DATA ACQUISITION

This program sets up and performs a data acquisition using High Speed Mode 4. It samples channels 4, 5, 6, and 7. The complete channel list is scanned 20 times. You should provide known signals on at least some of these channels to compare with the measured values.

Instructions used: **HS.RUN (Run High Speed)**
CNF.HS (Configure High Speed)
CNF.RG (Configure Rate Generator).

A.3.4 SAMPLE PROGRAM 4 DESCRIPTION: DMA DATA ACQUISITION

This program demonstrates DMA input running in DMA Mode 3. You must configure your Board to enable DMA on DMA channel 1. When you load PCI92S_Q.EXE, you must specify the switch **/B1**. This DMA mode requires a trigger signal, which you should supply on the External Interrupt input. The program samples channels 4, 5, 6, and 7. The complete channel list is scanned up to 64 times. You should provide known signals on at least some of these channels to compare with the measured values.

Instructions used: **CNF.DMA (Configure DMA)**
CNF.RG (Configure Rate Generator)
WRITE.CH (Write Channel)
DMA.RUN (Run DMA)
DMA.STAT (DMA Status)
BUF.DEC (Decode Buffer)
DMA.STOP (Stop DMA).

A.3.5 SAMPLE PROGRAM 5 DESCRIPTION: DMA DATA ACQUISITION

This program demonstrates DMA input running in DMA Mode 2. You must configure your Board to enable DMA on DMA channel 1. When you load PCI92S_Q.EXE, you must specify the switch **/B1**. The program samples channels 4, 5, 6, and 7. The complete channel list is scanned 20 times. You should provide

known signals on at least some of these channels to compare with the measured values.

Instructions used: **CNF.DMA (Configure DMA)**
CNF.RG (Configure Rate Generator)
WRITE.CH (Write Channel)
DMA.RUN (Run DMA)
DMA.STAT (DMA Status)
BUF.DEC (Decode Buffer)
DMA.STOP (Stop DMA).

A.4 Running the Sample Programs

All of the Sample Programs are written for Microsoft QuickBASIC and have file names with the extension .BAS, and all of them are ASCII files. This means that you can list them to the screen using the DOS TYPE command, or list them to the printer using the DOS PRINT command. You may also edit these files using a general-purpose editor.

The header file PCIHEADQ.BAS contains definitions and program statements which must be executed at the beginning of any program which accesses the Software Drivers. The Sample Programs incorporate the header file with the $INCLUDE metacommand. We recommend that you do this in your application programs as well. The Sample Programs also include an error-testing subroutine ErrorRoutine, which you may wish to copy into your application programs.

The Sample Programs are written for a PCI-20091W-1 Board with the address switches configured for a base address of CDC00 (Hex), the factory default setting. If you change this setting, you must alter the Sample Programs. Change the line

SEGMT = &HCDC0

to the segment part of the base address you have configured. For example, if you have set the base address to C8000, this line should read

SEGMT = &HC800

To run the Sample Programs, follow these steps:

1. Configure your PCI-20091W-1 Board and install it in your computer, as described in Chapter 2. Note that some of the sample programs expect specific jumper configurations.
2. Make a work disk or set up a hard disk subdirectory as explained in Chapter 3, and put your original Software Drivers diskette away in a safe place.
3. Load the Software Drivers by running PCI92S_Q.EXE. Specify the **/B1** switch if you will be running Sample Progams 4 or 5. Then load QuickBASIC with the User Library.

QB /L

(If the User Library is not provided with your version of QuickBASIC, refer to your QuickBASIC manual for instructions on how to create it.)

4. Determine which Sample Program you would like to try and load it into the QuickBASIC programming environment. If necessary, change the value of SEGMT, as described above.

5. Run the Sample Program, providing inputs and monitoring outputs, as appropriate. Feel free to modify the Sample Programs to experiment with your system.

A.5 Graphics Demonstration Program

The file **GRAPH.BAS** is a demonstration program that shows how to use three subroutines to build display windows and graph data in them.

The subroutine **InitWindow** sets up the coordinates of nine windows: four quarter-screen windows, four half-screen windows, and a full-screen window. These windows allow you to display up to four data sets simultaneously, or to use larger windows for one or two data sets.

The subroutine **PlotWindow** draws a graph box with tic marks for the specified window.

The subroutine **PlotData** scales a data set, calls **PlotWindow** to draw a box, labels the axes, and plots the data.

You can generate simple graphical displays of your data by incorporating these subroutines (or modified versions of them) into your application programs. The comments in **GRAPH.BAS** will show you how to use the subroutines.

Sample Program #1

```
'*************************************************************************
'*       PROGRAM NAME:    SAMPLE PROGRAM #1
'*       VERSION:         1.1
'*
'*       LANGUAGE:        BASIC
'*       TRANSLATOR:      MICROSOFT QUICK BASIC VERSION 4.0
'*************************************************************************

DECLARE SUB ErrorRoutine (ErrorString AS STRING)

'THIS PROGRAM READS AN ANALOG INPUT WITH A PCI-20091W-1 AND A
'PCI-20010T-1 USING THE FOLLOWING PCI-20092S-1 INSTRUCTIONS :

'       SYSINIT          INIT            ERR.SYS
'       READ.CH

'THE FOLLOWING HARDWARE CONFIGURATION IS REQUIRED:

'+-------------------------------------------------------------------+
'|                      PCI-20091W-1                                 |
'| +-----------------------------------------------+  +----------+  |
'| |  W6  - IN    W9  - IN    W15 - IN    W19 - IN |  |  &HCDC0  |  |
'| |                                               |  |----------|  |
'| |                                               |  | ADDRESS  |  |
'| |                                               |  +----------+  |
'| |                                               |  +----------+  |
'| |                                               |  | TIMEBASE |  |
'| |                                               |  +----------+  |
'| |                                               |  +----------+  |
'| |          ( ALL OTHER JUMPERS OUT )            |  | DMA H/W  |  |
'| +-----------------------------------------------+  +----------+  |
'+-------------------------------------+                +----------+
'                                      |_|_|_|_|_|_|_|_|

'THE ANALOG SIGNAL TO BE MEASURED MUST BE CONNECTED TO THE TERMINAL
'LABELED 0 ON THE PCI-20010T-1 AND THE PCI-20091W-1 MUST BE CONNECTED
'TO THE PCI-20010T-1.
'*************************************************************************

'  INCLUDE THE DRIVER INTERFACE HEADER FILE

'$INCLUDE: 'PCIHEADQ.BAS'

'  INITIALIZE THE PCI-20092S-1 SYSTEM.  THIS CALL MUST BE MADE PRIOR
'  TO CALLING ANY OTHER PCI-20092S-1 INSTRUCTION.

PRINT ""
PRINT "Begin PCI-20092S-1 Sample Program 1"

CALLS ABSOLUTE(SYSINIT)

'  SEGMT, DEFINED BELOW, IS THE BASE ADDRESS OF THE PCI-20091-1.  WE
'  USE &HCDC0, BUT YOU CAN SET IT ANYWHERE.  REFER TO THE PCI-20091-1
'  MANUAL FOR MORE INFORMATION.  INIT MUST BE CALLED ONCE FOR EACH
'  CARD IN THE SYSTEM, EACH WITH ITS OWN ADDRESS.

SEGMT = &HCDC0
CALLS ABSOLUTE(SEGMT, INIT)

'  CHECK FOR A SYSTEM ERROR.  DURING DEBUG, CALL ERR.SYS OFTEN,
'  POSSIBLY AFTER EVERY CALL TO THE PCI-20092S-1.  IT IS A GOOD IDEA
'  TO LEAVE A FEW ERR.SYS CALLS IN A FINISHED PROGRAM TO MONITOR THE
'  STATUS OF THE SYSTEM.

ErrorRoutine "Error found during INIT"

'  READ THE ANALOG INPUT CHANNEL.
'  PRE-DEFINE THE RETURNED VALUE ADATA.

ADATA = 0

PRINT "Reading data on 8 channels."
PRINT "This program assumes the board is configured for +/-10 V."
PRINT "Press any key to continue; 'Q' to stop."
PRINT

FOR CHN = 0 TO 7

    CALLS ABSOLUTE(AI.T, CHN, ADATA, READ.CH)
    ErrorRoutine "Error found during READ.CH"

```

```
82    '  CONVERT THE ADC READING, ADATA, INTO VOLTS AND DISPLAY.
83
84    '  THE ASSUMPTION IS THAT THE PCI-20091W-1 IS SET UP FOR +/-10 VOLTS.
85
86        FDATA = ((4095! - ADATA) * 20! / 4096! - 10!)
87
88    '  THE FOLLOWING TABLE GIVES THE EQUATIONS TO BE USED IN PLACE OF THE
89    '  ABOVE LINE FOR OTHER THAN +/-10 VOLTS.
90
91    '     +/- 2.5  FDATA = ((4095! - ADATA) *  5! / 4096! - 2.5!)
92    '     +/- 5    FDATA = ((4095! - ADATA) * 10! / 4096! -  5!)
93    '     0 TO 5   FDATA = ((4095! - ADATA) *  5! / 4096!       )
94    '     0 TO 10  FDATA = ((4095! - ADATA) * 10! / 4096!       )
95
96        PRINT "Channel "; CHN; " Analog Input voltage is ";
97        PRINT USING "###.##"; FDATA
98
99    '  WAIT FOR A KEY THEN TAKE ANOTHER READING. PRESS 'Q' TO STOP.
100
101       DO
102           K$ = INKEY$
103       LOOP WHILE K$ = ""
104
105       IF K$ = "Q" OR K$ = "q" THEN END
106
107   NEXT CHN
108
109   PRINT "Done"
110
111   END
112
113   DEFSNG A-E, G-Z
114   SUB ErrorRoutine (ErrorString AS STRING)
115
116   '  DEFINE A GENERAL PURPOSE ERROR PROCESSING ROUTINE.
117
118     ERROR.CODE% = 0
119     CALLS ABSOLUTE(ERROR.CODE%, ERR.SYS)
120
121     IF ERROR.CODE% <> 0 THEN
122
123       PRINT ErrorString; ERROR.CODE%
124       END
125
126     END IF
127
128   END SUB
129
```

Sample Program #2

```
1     '************************************************************************
2     '*       PROGRAM NAME:   SAMPLE PROGRAM #2
3     '*       VERSION:        1.1
4     '*
5     '*       LANGUAGE:       BASIC
6     '*       TRANSLATOR:     MICROSOFT QUICK BASIC VERSION 4.0
7     '************************************************************************
8
9     DECLARE SUB ErrorRoutine (ErrorString AS STRING)
10
11    'THIS PROGRAM OUTPUTS A PULSE TRAIN FROM THE RATE GENERATOR ON THE
12    'PCI-20091W-1 USING THE FOLLOWING PCI-20092S-1 INSTRUCTIONS:
13
14    '       SYSINIT         INIT            ERR.SYS
15    '       CNF.RG          WRITE.CH
16
17    'THE FOLLOWING HARDWARE CONFIGURATION IS REQUIRED:
18
19    '+-------------------------------------------------------------------+
20    '|                      PCI-20091W-1                                 |
21    '| +-----------------------------------------------+ +----------+  |
22    '| |                                               | |  &HCDC0  |  |
23    '| |  ( JUMPER SETTINGS DO NOT AFFECT THE RATE     | |----------|  |
24    '| |    GENERATOR.  THE FACTORY CONFIGURATION IS    | | ADDRESS  |  |
25    '| |    FINE. }                                    | +----------+  |
26    '| |                                               | +----------+  |
27    '| |                                               | | TIMEBASE |  |
28    '| |                                               | +----------+  |
29    '| |                                               | +----------+  |
30    '| |                                               | | DMA H/W  |  |
31    '| +-----------------------------------------------+ +----------+  |
32    '+-------------------------------------------+         +----------+
```

```
33  '                                          |_|_|_|_|_|_|_|_|
34
35  'THE TTL DIGITAL PULSE TRAIN GENERATED BY THIS PROGRAM IS AVAILABLE
36  'ON CONNECTOR P3 PIN 1 OF THE PCI-20091W-1 AND DIGITAL GROUND IS
37  'AVAILABLE ON CONNECTOR P3 PIN 6 OF THE PCI-20091W-1.
38  '*************************************************************************
39
40  '  INCLUDE THE DRIVER INTERFACE HEADER FILE
41
42  '$INCLUDE: 'PCIHEADQ.BAS'
43
44  '  INITIALIZE THE PCI-20092S-1 SYSTEM.  THIS CALL MUST BE MADE PRIOR
45  '  TO CALLING ANY OTHER PCI-20092S-1 INSTRUCTION.
46
47  PRINT ""
48  PRINT "Begin PCI-20092S-1 Sample Program 2"
49
50  CALLS ABSOLUTE(SYSINIT)
51
52  '  SEGMT, DEFINED BELOW, IS THE BASE ADDRESS OF THE PCI-20091-1.  WE
53  '  USE &HCDC0, BUT YOU CAN SET IT ANYWHERE.  REFER TO THE PCI-20091-1
54  '  MANUAL FOR MORE INFORMATION.  INIT MUST BE CALLED ONCE FOR EACH
55  '  CARD IN THE SYSTEM, EACH WITH ITS OWN ADDRESS.
56
57  SEGMT = &HCDC0
58  CALLS ABSOLUTE(SEGMT, INIT)
59
60  '  CHECK FOR A SYSTEM ERROR.  DURING DEBUG, CALL ERR.SYS OFTEN,
61  '  POSSIBLY AFTER EVERY CALL TO THE PCI-20092S-1.  IT IS A GOOD IDEA
62  '  TO LEAVE A FEW ERR.SYS CALLS IN A FINISHED PROGRAM TO MONITOR THE
63  '  STATUS OF THE SYSTEM.
64
65  ErrorRoutine "Error found during INIT"
66
67  '  CONFIGURE RATE GENERATOR CHANNEL 0 WITH COUNTER ONE SET TO 4,
68  '  COUNTER TWO SET TO 2, AND MODE 2.
69
70  CHN = 0
71  CNT1 = 4
72  CNT2 = 2
73  MODE = 2
74  CALLS ABSOLUTE(CHN, CNT1, CNT2, MODE, CNF.RG)
75  ErrorRoutine "Error found during CNF.RG"
76
77  '  ALTERNATELY ENABLE AND DISABLE THE RATE GENERATOR.
78
79  PRINT "The Rate Generator has been configured to generate a square wave."
80  PRINT "The program will now enable or disable the Rate Generator whenever"
81  PRINT "a key is pressed.  Press 'Q' to quit."
82  PRINT
83
84  ENABLE = 1
85  DISABLE = 0
86  RGDATA = ENABLE
87
88  DO
89      CALLS ABSOLUTE(RG.T, CHN, RGDATA, WRITE.CH)
90
91      IF RGDATA = ENABLE THEN
92          ErrorRoutine "Error found during WRITE.CH (enable)"
93          PRINT "ON ";
94          RGDATA = DISABLE
95      ELSE
96          ErrorRoutine "Error found during WRITE.CH (disable)"
97          PRINT "off ";
98          RGDATA = ENABLE
99      END IF
100
101 '  WAIT FOR A KEY THEN CHANGE THE RATE GENERATOR STATE.
102
103     DO
104         K$ = INKEY$
105     LOOP WHILE K$ = ""
106
107 LOOP WHILE (K$ <> "Q" AND K$ <> "q")
108
109 PRINT
110 PRINT "Done."
111
112 END
113
114 DEFSNG A-E, G-Z
115 SUB ErrorRoutine (ErrorString AS STRING)
116
117 '  DEFINE A GENERAL PURPOSE ERROR PROCESSING ROUTINE.
118
119   ERROR.CODE% = 0
```

```
120     CALLS ABSOLUTE(ERROR.CODE%, ERR.SYS)
121
122     IF ERROR.CODE% <> 0 THEN
123
124       PRINT ErrorString; ERROR.CODE%
125       END
126
127     END IF
128
129   END SUB
130
```

Sample Program #3

```
1   '*********************************************************************
2   '*       PROGRAM NAME:   SAMPLE PROGRAM #3
3   '*       VERSION:        1.1
4   '*
5   '*       LANGUAGE:       BASIC
6   '*       TRANSLATOR:     MICROSOFT QUICK BASIC VERSION 4.0
7   '*********************************************************************
8
9   DECLARE SUB ErrorRoutine (ErrorString AS STRING)
10
11  'THIS PROGRAM DEMONSTRATES THE HIGH SPEED FEATURE OF THE PCI-20091W-1
12  'WITH A PCI-20010T-1 USING THE FOLLOWING PCI-20092S-1 INSTRUCTIONS:
13
14  '       SYSINIT         INIT            ERR.SYS
15  '       HS.RUN          CNF.HS          CNF.RG
16
17  'THE FOLLOWING HARDWARE CONFIGURATION IS REQUIRED :
18
19  '+-------------------------------------------------------------+
20  '|                    PCI-20091W-1                             |
21  '|  +-----------------------------------------+  +----------+  |
22  '|  |  W6  - IN    W9  - IN    W12 - IN    W14 - IN  |  | &HCDC0   |  |
23  '|  |                                         |  |----------|  |
24  '|  |                                         |  | ADDRESS  |  |
25  '|  |                                         |  +----------+  |
26  '|  |                                         |  +----------+  |
27  '|  |                                         |  | TIMEBASE |  |
28  '|  |                                         |  +----------+  |
29  '|  |                                         |  +----------+  |
30  '|  |          ( ALL OTHER JUMPERS OUT )      |  | DMA H/W  |  |
31  '|  +-----------------------------------------+  +----------+  |
32  '+-------------------------------------+          +----------+
33  '                                      |_|_|_|_|_|_|_|_|
34
35  'THE ANALOG INPUT SIGNALS MUST BE CONNECTED TO THE TERMINALS LABELED
36  '4, 5, 6, AND 7 ON THE PCI-20010T-1 AND THE PCI-20010T-1 MUST BE
37  'CONNECTED TO THE PCI-20091W-1.
38  '*********************************************************************
39
40  '  INCLUDE THE DRIVER INTERFACE HEADER FILE
41
42  '$INCLUDE: 'PCIHEADQ.BAS'
43
44  '  INITIALIZE THE PCI-20092S-1 SYSTEM.  THIS CALL MUST BE MADE PRIOR
45  '  TO CALLING ANY OTHER PCI-20092S-1.
46
47  PRINT ""
48  PRINT "Begin PCI-20092S-1 Sample Program 3"
49
50  CALLS ABSOLUTE(SYSINIT)
51
52  '  SEGMT, DEFINED BELOW, IS THE BASE ADDRESS OF THE PCI-20091-1.  WE
53  '  USE &HCDC0, BUT YOU CAN SET IT ANYWHERE.  REFER TO THE PCI-20091-1
54  '  MANUAL FOR MORE INFORMATION.  INIT MUST BE CALLED ONCE FOR EACH
55  '  CARD IN THE SYSTEM, EACH WITH ITS OWN ADDRESS.
56
57  SEGMT = &HCDC0
58  CALLS ABSOLUTE(SEGMT, INIT)
59
60  '  CHECK FOR A SYSTEM ERROR.  DURING DEBUG, CALL ERR.SYS OFTEN,
61  '  POSSIBLY AFTER EVERY CALL TO THE PCI-20092S-1.  IT IS A GOOD IDEA
62  '  TO LEAVE A FEW ERR.SYS CALLS IN A FINISHED PROGRAM TO MONITOR THE
63  '  STATUS OF THE SYSTEM.
64
65  ErrorRoutine "Error found during INIT"
66
67  '  CONFIGURE THE SYSTEM FOR HIGH SPEED RUN.  USE RATE GENERATOR
68  '  CHANNEL 0 AS THE PACER IN MODE 4.
69  '  READ ANALOG INPUT CHANNELS 4, 5, 6, AND 7.
```

```
70
71    PRINT "Configuring High Speed Run ... ";
72    DIM HSCHNS(5)
73    PAC.TYPE = RG.T
74    PAC.CHN = 0
75    MODE = 4
76    TR.TYPE = 0
77    TR.CHN = 0
78    HSCHNS(1) = 4: HSCHNS(2) = 5: HSCHNS(3) = 6
79    HSCHNS(4) = 7: HSCHNS(5) = -1
80    CALLS ABSOLUTE(PAC.TYPE, PAC.CHN, MODE, TR.TYPE, TR.CHN, HSCHNS(1), CNF.HS)
81    ErrorRoutine "Error found during CNF.HS"
82
83    PRINT "Done"
84
85    '  CONFIGURE THE RATE GENERATOR PACER FREQUENCY.
86
87    PRINT "Configuring Rate Generator ... ";
88    RGCHN = 0
89    COUNT1 = 1000
90    COUNT2 = 2
91    MODE = 2
92    CALLS ABSOLUTE(RGCHN, COUNT1, COUNT2, MODE, CNF.RG)
93    ErrorRoutine "Error found during CNF.RG"
94
95    PRINT "Done"
96
97    '  START THE HIGH SPEED ACQUISITION
98
99    PRINT
100   PRINT "Ready to perform High Speed Acquisition.  Press 'G' to start... ";
101
102   DO
103       K$ = INKEY$
104   LOOP WHILE K$ = ""
105
106   PRINT
107
108   IF K$ <> "G" AND K$ <> "g" THEN
109       PRINT "High Speed Acquisition cancelled."
110       END
111   END IF
112
113   PRINT "Start High Speed Acquisition ... ";
114   PASS = 20
115   DIM DARRAY(80)
116   SEGMNT = 0
117   CALLS ABSOLUTE(PASS, DARRAY(1), SEGMNT, HS.RUN)
118   ErrorRoutine "Error found during HS.RUN"
119
120   PRINT "Done"
121
122   '  DISPLAY THE COLLECTED DATA
123
124   PRINT "                      Channel"
125   PRINT " Pass";
126
127   FOR J = 1 TO 4
128       PRINT "      "; HSCHNS(J);
129   NEXT J
130
131   PRINT
132   PRINT
133
134   FOR I = 1 TO PASS
135       PRINT USING "  ##    "; I;
136
137       FOR J = 1 TO 4
138           FDATA = ((4095! - DARRAY(4 * (I - 1) + J)) / 4096! * 20!) - 10!
139           PRINT USING "###.### "; FDATA;
140       NEXT J
141       PRINT
142   NEXT I
143
144   END
145
146   DEFSNG A-E, G-Z
147   SUB ErrorRoutine (ErrorString AS STRING)
148
149   '  DEFINE A GENERAL PURPOSE ERROR PROCESSING ROUTINE.
150
151     ERROR.CODE% = 0
152     CALLS ABSOLUTE(ERROR.CODE%, ERR.SYS)
153
154     IF ERROR.CODE% <> 0 THEN
155
156       PRINT ErrorString; ERROR.CODE%
```

```
157       END
158
159     END IF
160
161   END SUB
```

Sample Program #4

```
1   '*************************************************************************
2   '*       PROGRAM NAME:    SAMPLE PROGRAM #4
3   '*       VERSION:         1.1
4   '*
5   '*       LANGUAGE:        BASIC
6   '*       TRANSLATOR:      MICROSOFT QUICK BASIC VERSION 4.0
7   '*************************************************************************
8
9   DECLARE SUB ErrorCheck (ErrorString AS STRING)
10
11  'THIS PROGRAM DEMONSTRATES THE DMA INPUT FEATURE OF THE PCI-20091W-1,
12  'RUNNING IN DMA MODE 3, WITH A PCI-20010T-1. WHEN PCI-20092S-1 IS
13  'LOADED, USE /B1 TO LIMIT THE DMA BUFFER TO 1024 BYTES. OTHERWISE THE
14  'AMOUNT OF DATA TAKEN MAY BE UNWIELDY. THE PROGRAM USES THE FOLLOWING
15  'PCI-20092S-1 INSTRUCTIONS:
16
17  '       SYSINIT          INIT             ERR.SYS
18  '       CNF.DMA          CNF.RG           WRITE.CH        DMA.RUN
19  '       DMA.STAT         BUF.DEC          DMA.STOP
20
21  'THE FOLLOWING HARDWARE CONFIGURATION IS REQUIRED :
22
23  '+-------------------------------------------------------------------+
24  '|                      PCI-20091W-1                                 |
25  '|  +-----------------------------------------------+  +----------+  |
26  '|  |  W6  - IN    W9  - IN    W12 - IN    W14 - IN |  |  &HCDC0  |  |
27  '|  |  W27 - IN    W28 - IN                         |  |----------|  |
28  '|  |                                               |  | ADDRESS  |  |
29  '|  |                                               |  +----------+  |
30  '|  |                                               |  +----------+  |
31  '|  |                                               |  | TIMEBASE |  |
32  '|  |                                               |  +----------+  |
33  '|  |                                               |  +----------+  |
34  '|  |          ( ALL OTHER JUMPERS OUT )            |  | DMA H/W  |  |
35  '|  +-----------------------------------------------+  +----------+  |
36  '+-----------------------------------------+              +----------+
37  '                                          |_|_|_|_|_|_|_|_|
38
39  'THE ANALOG INPUT SIGNALS MUST BE CONNECTED TO THE TERMINALS LABELED
40  '4, 5, 6, AND 7 ON THE PCI-20010T-1 AND THE PCI-20010T-1 MUST BE
41  'CONNECTED TO THE PCI-20091W-1.
42  '*************************************************************************
43
44  '  INCLUDE THE DRIVER INTERFACE HEADER FILE
45
46  '$INCLUDE: 'PCIHEADQ.BAS'
47
48  '  INITIALIZE THE PCI-20092S-1 SYSTEM.  THIS CALL MUST BE MADE PRIOR
49  '  TO CALLING ANY OTHER PCI-20092S-1 INSTRUCTION.
50
51  PRINT ""
52  PRINT "Begin PCI-20092S-1 Sample Program 4"
53
54  CALLS ABSOLUTE(SYSINIT)
55
56  '  SEGMT, DEFINED BELOW, IS THE BASE ADDRESS OF THE PCI-20091-1.  WE
57  '  USE &HCDC0, BUT YOU CAN SET IT ANYWHERE.  REFER TO THE PCI-20091-1
58  '  MANUAL FOR MORE INFORMATION.  INIT MUST BE CALLED ONCE FOR EACH
59  '  CARD IN THE SYSTEM, EACH WITH ITS OWN ADDRESS.
60
61  SEGMT = &HCDC0
62  CALLS ABSOLUTE(SEGMT, INIT)
63
64  '  CHECK FOR A SYSTEM ERROR.  DURING DEBUG, CALL ERR.SYS OFTEN,
65  '  POSSIBLY AFTER EVERY CALL TO THE PCI-20092S-1.  IT IS A GOOD IDEA
66  '  TO LEAVE A FEW ERR.SYS CALLS IN A FINISHED PROGRAM TO MONITOR THE
67  '  STATUS OF THE SYSTEM.
68
69  ErrorCheck "Error found during INIT"
70
71  '  CONFIGURE THE SYSTEM FOR DMA INPUT.  USE THE END-OF-CONVERT SIGNAL
72  '  FROM THE ANALOG INPUT SECTION AS THE PACER, TRIGGER OFF OF THE
73  '  XSYNC INPUT TO THE BOARD (CONNECTOR P3 PIN 5), RUN IN DMA MODE 3,
74  '  AND ACQUIRE ANALOG CHANNELS 4, 5, 6, AND 7.
75
```

```
   76   PRINT "Configuring DMA ... ";
   77   DIM CHNS(5): DIM TYPES(4)
   78   PAC.TYPE = AI.T
   79   PAC.CHN = 0
   80   DMODE = 3
   81   TR.TYPE = XSYNC.T
   82   TR.CHN = 0
   83
   84   TYPES(1) = AI.T:  CHNS(1) = 4
   85   TYPES(2) = AIX.T: CHNS(2) = 5
   86   TYPES(3) = AIX.T: CHNS(3) = 6
   87   TYPES(4) = AIX.T: CHNS(4) = 7
   88                     CHNS(5) = -1          ' END OF LIST
   89
   90   LENGTH = 0
   91   CALLS ABSOLUTE(PAC.TYPE, PAC.CHN, DMODE, TR.TYPE, TR.CHN, TYPES(1), CHNS(1), LENGTH,
CNF.DMA)
   92   ErrorCheck "Error found during CNF.DMA"
   93
   94   PRINT "Done"
   95
   96   '  CONFIGURE THE RATE GENERATOR FREQUENCY.  THE RATE GENERATOR OUTPUT
   97   '  SIGNAL WILL START ANALOG-TO-DIGITAL CONVERSIONS.  WHEN EACH
   98   '  CONVERSION IS COMPLETE, THE PCI-20091W-1 WILL GENERATE THE DMA
   99   '  PACER SIGNAL (END-OF-CONVERT).
  100
  101   COUNT1 = 100
  102   COUNT2 = 2
  103   RMODE = 2
  104   RG.CHN = 0
  105   PRINT "Configuring Rate Generator ... ";
  106   CALLS ABSOLUTE(RG.CHN, COUNT1, COUNT2, RMODE, CNF.RG)
  107   ErrorCheck "Error found during CNF.RG"
  108
  109   PRINT "Done"
  110
  111   '  SET UP THE DMA ACQUISITION
  112
  113   SFRAMES = 0
  114   DELAY = 20
  115   PRINT "Setting Up DMA Acquisition ... ";
  116   CALLS ABSOLUTE(SFRAMES, DELAY, DMA.RUN)
  117   ErrorCheck "Error found during DMA.RUN"
  118
  119   PRINT "Done"
  120
  121   '  START THE ACQUISITION
  122
  123   ENABLE = 1
  124   PRINT "Starting DMA Acquisition ... ";
  125   CALLS ABSOLUTE(RG.T, RG.CHN, ENABLE, WRITE.CH)
  126   ErrorCheck "Error found during WRITE.CH"
  127
  128   PRINT "Done"
  129
  130   '  WAIT FOR "TRIGGER AND DELAY" TO COMPLETE.
  131   '  PRE-DEFINE NUMCLSTS (NUMBER OF CLUSTERS READ) AND STATUS.
  132
  133   PRINT
  134   PRINT "DMA running.  Press any key to abort."
  135   PRINT "Waiting for Trigger and Delay to Complete ... ";
  136   NUMCLSTS = 0
  137   STATUS = 0
  138
  139   DO
  140       CALLS ABSOLUTE(NUMCLSTS, STATUS, DMA.STAT)
  141       ErrorCheck "Error found during DMA.STAT (trigger and delay wait)"
  142       K$ = INKEY$
  143   LOOP WHILE STATUS = 3 AND K$ = ""
  144
  145   IF STATUS = 3 THEN
  146       PRINT "Aborted."
  147       CALLS ABSOLUTE(NUMCLSTS, DMA.STOP)
  148       ERR.CODE = 0
  149       CALLS ABSOLUTE(ERR.CODE, ERR.SYS)
  150       IF ERR.CODE <> 0 THEN
  151           PRINT "Error found during DMA.STOP "; ERR.CODE; ", continuing"
  152       END IF
  153   ELSE
  154       PRINT "Done."
  155   END IF
  156
  157   ' THE ACQUISITION IS NOW COMPLETE
  158
  159   PRINT
  160   PRINT NUMCLSTS; " clusters of data are available."
  161   PRINT "Press any character to display next cluster; 'Q' to quit."
```

```
162
163   ' PRINTS THE DATA IN THE BUFFER.  USE THE BUF.DEC() INSTRUCTION TO
164   ' ACCESS THE DATA IN THE BUFFER.
165
166   DIM CLUST(4)
167
168   PRINT
169   PRINT "                              Channel"
170   PRINT "Cluster";
171
172   FOR J = 1 TO 4
173       PRINT USING "        #"; CHNS(J);
174   NEXT J
175
176   PRINT
177
178   FOR I = 1 TO NUMCLSTS
179
180       DEC.CLUSTS = 1
181       NUM.DECODED = 0
182
183       CALLS ABSOLUTE(DEC.CLUSTS, CLUST(1), NUM.DECODED, BUF.DEC)
184       ErrorCheck "Error found while decoding buffer data"
185
186       IF (NUM.DECODED <> DEC.CLUSTS) THEN
187           PRINT "Internal Error -- Did not decode full cluster"
188           END
189       END IF
190
191   ' PRINT THE DATA IN ALL FOUR FRAMES OF THE CLUSTER FIELD-BY-FIELD.
192
193       PRINT USING "  ##    "; I;
194
195       FOR J = 1 TO 4
196           FDATA = (20! / 4096!) * (4095! - CLUST(J)) - 10!
197           PRINT USING "   ###.##"; FDATA;
198       NEXT J
199
200       DO
201           K$ = INKEY$
202       LOOP WHILE K$ = ""
203
204       PRINT
205       IF K$ = "Q" OR K$ = "q" THEN END
206
207   NEXT I
208
209   PRINT "Done."
210
211   END
212
213   DEFSNG A-E, G-Z
214   SUB ErrorCheck (ErrorString AS STRING)
215
216   '  DEFINE A GENERAL PURPOSE ERROR PROCESSING ROUTINE.
217
218     ERROR.CODE% = 0
219     CALLS ABSOLUTE(ERROR.CODE%, ERR.SYS)
220
221     IF ERROR.CODE% <> 0 THEN
222
223   '   STOP ANY DMA ACQUISITION THAT MIGHT HAVE BEEN RUNNING WHEN THE
224   '   ERROR OCCURRED.
225
226        CLUSTS% = 0
227        CALLS ABSOLUTE(CLUSTS%, DMA.STOP)
228
229   '   PRINT THE ERROR MESSAGE PASSED IN AS A PARAMETER AND THEN PRINT THE
230   '   ERROR CODE NUMBER
231
232        PRINT ErrorString; ERROR.CODE%
233        END
234
235     END IF
236
237   END SUB
238
```

Sample Program #5

```
1   '*************************************************************************
2   '*       PROGRAM NAME:    SAMPLE PROGRAM #5
3   '*       VERSION:         1.1
```

```
4   '*
5   '*       LANGUAGE:       BASIC
6   '*       TRANSLATOR:     MICROSOFT QUICK BASIC VERSION 4.0
7   '*************************************************************************
8
9   DECLARE SUB ErrorCheck (ErrorString AS STRING)
10
11  'THIS PROGRAM DEMONSTRATES THE DMA INPUT FEATURE OF THE PCI-20091W-1,
12  'RUNNING IN DMA MODE 2, WITH A PCI-20010T-1. WHEN PCI-20092S-1 IS
13  'LOADED, USE /B TO GET A 64K DMA BUFFER.  THE PROGRAM USES THE
14  'FOLLOWING PCI-20026S-1 INSTRUCTIONS:
15
16  '       SYSINIT         INIT            ERR.SYS
17  '       BUF.DEC         CNF.DMA         CNF.RG          DMA.RUN
18  '       DMA.STAT        DMA.STOP        WRITE.CH
19
20  'THE FOLLOWING HARDWARE CONFIGURATION IS REQUIRED :
21
22  '+-------------------------------------------------------------------+
23  '|                      PCI-20091W-1                                 |
24  '| +-----------------------------------------------+  +----------+  |
25  '| |  W6  - IN    W9  - IN    W12 - IN    W14 - IN |  |  &HCDC0  |  |
26  '| |  W27 - IN    W28 - IN                         |  |----------|  |
27  '| |                                               |  | ADDRESS  |  |
28  '| |                                               |  +----------+  |
29  '| |                                               |  +----------+  |
30  '| |                                               |  | TIMEBASE |  |
31  '| |                                               |  +----------+  |
32  '| |                                               |  +----------+  |
33  '| |          ( ALL OTHER JUMPERS OUT )            |  | DMA H/W  |  |
34  '| +-----------------------------------------------+  +----------+  |
35  '+-------------------------------------+                +----------+
36  '                                      |_|_|_|_|_|_|_|_|
37
38  'THE ANALOG INPUT SIGNALS MUST BE CONNECTED TO THE TERMINALS LABELED
39  '4, 5, 6, AND 7 ON THE PCI-20010T-1 AND THE PCI-20010T-1 MUST BE
40  'CONNECTED TO THE PCI-20091W-1.
41  '*************************************************************************
42
43  '  INCLUDE THE DRIVER INTERFACE HEADER FILE
44
45  '$INCLUDE: 'PCIHEADQ.BAS'
46
47  '  DEFINE DMA DATA ACQUISITION PARAMETERS
48
49  FRAMES.PER.CLUSTER = 4          ' NUMBER OF FRAMES PER CLUSTER
50  BOARD.SEG = &HCDC0              ' BOARD SEGMENT ADDRESS
51
52  PAC.TYPE = AI.T                 ' PACER TYPE AND CHANNEL
53  PAC.CHN = 0
54
55  TRIG.TYPE = 0                   ' TRIGGER TYPE AND CHANNEL (NO TRIGGER)
56  TRIG.CHN = 0
57
58  DMA.MODE = 2                    ' DMA MODE FOR THE ACQUISITION
59  TERMINAL.COUNT = 20             ' DESIRED NUMBER OF CLUSTERS
60
61  '  DEFINE THE DMA CHANNEL AND TYPE LISTS.  THESE LISTS SPECIFY THE
62  '  CHANNELS WHICH WILL BE READ BY DMA IN EACH CLUSTER.  THE -1 ELEMENT
63  '  IN THE CHANNELS LIST IS A MARKER OF THE END OF THE LIST.
64
65  DIM TYPES(4)
66  DIM CHNS(5)
67
68  TYPES(1) = AI.T:  CHNS(1) = 4
69  TYPES(2) = AIX.T: CHNS(2) = 5
70  TYPES(3) = AIX.T: CHNS(3) = 6
71  TYPES(4) = AIX.T: CHNS(4) = 7
72                    CHNS(5) = -1
73
74  '  DEFINE RATE GENERATOR PARAMETERS
75
76  COUNT1 = 2400                   ' RATE GENERATOR COUNTS
77  COUNT2 = 2
78  RG.CHN = 0                      ' RATE GENERATOR CHANNEL
79  RG.MODE = 2                     ' RATE GENERATOR MODE
80  RG.ENABLE = 1                   ' RATE GENERATOR ENABLE VALUE
81  '*************************************************************************
82
83  '  INITIALIZE THE PCI-20092S-1 SYSTEM.  THIS CALL MUST BE MADE PRIOR
84  '  TO CALLING ANY OTHER PCI-20092S-1 INSTRUCTION.
85
86  PRINT ""
87  PRINT "Begin PCI-20092S-1 Sample Program 5"
88
89  CALLS ABSOLUTE(SYSINIT)
90
```

```
91    '  SEGMT, DEFINED BELOW, IS THE BASE ADDRESS OF THE PCI-20091-1.  WE
92    '  USE &HCDC0, BUT YOU CAN SET IT ANYWHERE.  REFER TO THE PCI-20091-1
93    '  MANUAL FOR MORE INFORMATION.  INIT MUST BE CALLED ONCE FOR EACH
94    '  CARD IN THE SYSTEM, EACH WITH ITS OWN ADDRESS.
95
96    CALLS ABSOLUTE(BOARD.SEG, INIT)
97
98    '  CHECK FOR A SYSTEM ERROR.  DURING DEBUG, CALL ERR.SYS OFTEN,
99    '  POSSIBLY AFTER EVERY CALL TO THE PCI-20092S-1.  IT IS A GOOD IDEA
100   '  TO LEAVE A FEW ERR.SYS CALLS IN A FINISHED PROGRAM TO MONITOR THE
101   '  STATUS OF THE SYSTEM.
102
103   ErrorCheck "Error found during INIT"
104
105   '  CONFIGURE THE RATE GENERATOR CHANNEL.  THE CHANNEL NUMBER TO
106   '  CONFIGURE AND THE COUNT AND MODE CONFIGURATION PARAMETERS MAY BE
107   '  FOUND IN RG.CHN, COUNT1, COUNT2, AND RG.MODE.
108
109   PRINT "Configuring rate generator channel... ";
110   CALLS ABSOLUTE(RG.CHN, COUNT1, COUNT2, RG.MODE, CNF.RG)
111   ErrorCheck "Error found in call to CnfRg"
112   PRINT "Done."
113
114   '  CONFIGURE THE DMA PROCESS WHICH WILL BE RUN LATER.  THE PACER AND
115   '  TRIGGER TYPES AND CHANNELS ARE DEFINED IN PAC.TYPE, PAC.CHN,
116   '  TRIG.TYPE, AND TRIG.CHN.  THE DMA CHANNEL AND TYPE LISTS MAY BE
117   '  FOUND IN CHNS AND TYPES, RESPECTIVELY.  THE NUMBER OF BYTES IN EACH
118   '  CLUSTER IS RETURNED TO THE CALLER.
119
120   '  1) THE DMA PROCESS WILL BE PACED BY THE END-OF-CONVERSION SIGNAL
121   '     FROM THE PCI-20091W-1.  THIS SIGNAL BECOMES ACTIVE WHEN AN
122   '     ANALOG CHANNEL VALUE HAS BEEN CONVERTED TO A DIGITAL VALUE AND
123   '     IS READY TO BE READ.
124   '  2) THE START-CONVERT SIGNAL TO THE PCI-20091W-1 IS SUPPLIED FROM
125   '     THE ON-BOARD RATE GENERATOR.
126   '  3) THE DMA PROCESS WILL BEGIN ON COMMAND AND WILL STOP WHEN THE
127   '     TERMINAL COUNT IS REACHED.
128
129   PRINT "Configuring DMA... ";
130
131   '  PRE-DEFINE RETURNED VALUE BYTES.PER.CLUSTER
132
133   BYTES.PER.CLUSTER = 0
134   CALLS ABSOLUTE(PAC.TYPE, PAC.CHN, DMA.MODE, TRIG.TYPE, TRIG.CHN, TYPES(1), CHNS(1),
BYTES.PER.CLUSTER, CNF.DMA)
135   ErrorCheck "Error found during call to CNF.DMA"
136   PRINT "Done."
137
138   '  ACQUIRE DATA:  CALL DMA.RUN TO ENABLE THE DMA CONTROLLER AND START
139   '  THE MODE 2 ACQUISITION AND CALL WRITE.CH TO ENABLE THE RATE
140   '  GENERATOR.  WHEN THE DMA.RUN CALL RETURNS, CALL THE DMA.STAT
141   '  STATUS ROUTINE TO DETERMINE WHEN THE ACQUISITION IS COMPLETE.
142
143   '  RUN DMA ACQUISITION IN MODE 2:
144   '     THE SECOND PARAMETER, DELAY, IS IGNORED IN MODE 2
145
146   PRINT "Running DMA acquisition...";
147   DELAY = 0
148   CALLS ABSOLUTE(TERMINAL.COUNT, DELAY, DMA.RUN)
149   ErrorCheck "Error found on call to DMA.RUN"
150
151   '  ENABLE RATE GENERATOR TO SEND START-CONVERT SIGNALS.
152
153   PRINT "Starting rate generator/start-convert signal... ";
154   CALLS ABSOLUTE(RG.T, RG.CHN, RG.ENABLE, WRITE.CH)
155   ErrorCheck "Error encountered during WRITE.CH (to start RG)"
156   PRINT "Done."
157
158   '  WAIT FOR THE DMA PROCESS TO COMPLETE.
159   '  PRE-DEFINE THE RETURNED VALUES STAT AND CLUSTERS.
160
161   STAT = 0
162   CLUSTERS = 0
163
164   PRINT
165   PRINT "DMA process running.  Press any key to abort."
166   PRINT "Waiting for DMA process to complete... ";
167
168   DO
169       CALLS ABSOLUTE(CLUSTERS, STAT, DMA.STAT)
170       ErrorCheck "Error in call to DMA.STAT"
171       K$ = INKEY$
172   LOOP WHILE STAT = 2 AND K$ = ""
173
174   IF STAT = 2 THEN
175       PRINT "Aborted."
176       CALLS ABSOLUTE(CLUSTERS, DMA.STOP)
```

```
177       ERR.CODE = 0
178       CALLS ABSOLUTE(ERR.CODE, ERR.SYS)
179       IF ERR.CODE <> 0 THEN
180           PRINT "Error found during DMA.STOP "; ERR.CODE; ", continuing"
181       END IF
182   ELSE
183       PRINT "Done."
184   END IF
185
186   '  THE ACQUISITION IS NOW COMPLETE
187
188   PRINT
189   PRINT CLUSTERS; " clusters of data are available."
190   PRINT "Press any character to display next cluster; 'Q' to quit."
191
192   '  PRINT THE DATA IN THE BUFFER.
193
194   DIM CLUST(4)
195
196   PRINT
197   PRINT "                              Channel"
198   PRINT "Cluster";
199
200   FOR J = 1 TO 4
201       PRINT USING "         #"; CHNS(J);
202   NEXT J
203
204   PRINT
205   DEC.CLUSTS = 1
206
207   FOR I = 1 TO CLUSTERS
208
209   '  USE BUF.DEC TO READ AND PRINT EACH CLUSTER ELEMENT
210   '  PRE-DEFINE THE RETURNED VALUE NUM.DECODED.
211
212       NUM.DECODED = 0
213       CALLS ABSOLUTE(DEC.CLUSTS, CLUST(1), NUM.DECODED, BUF.DEC)
214       ErrorCheck "Error found while decoding buffer data"
215
216       IF (NUM.DECODED <> DEC.CLUSTS) THEN
217           PRINT "Internal Error -- Did not decode full cluster"
218           END
219       END IF
220
221   ' PRINT THE DATA IN ALL FOUR FRAMES OF THE CLUSTER FIELD-BY-FIELD.
222       PRINT USING "  ##    "; I;
223
224       FOR J = 1 TO 4
225           FDATA = (20! / 4096!) * (4095! - CLUST(J)) - 10!
226           PRINT USING "   ###.##"; FDATA;
227       NEXT J
228
229       PRINT
230
231       DO
232           K$ = INKEY$
233       LOOP WHILE K$ = ""
234
235       IF K$ = "Q" OR K$ = "q" THEN END
236
237   NEXT I
238
239   PRINT "Done."
240
241   END
242
243   DEFSNG A-E, G-Z
244   SUB ErrorCheck (ErrorString AS STRING)
245
246   '  DEFINE A GENERAL PURPOSE ERROR PROCESSING ROUTINE.
247
248     ERROR.CODE% = 0
249     CALLS ABSOLUTE(ERROR.CODE%, ERR.SYS)
250
251     IF ERROR.CODE% <> 0 THEN
252
253   '  STOP ANY DMA ACQUISITION THAT MIGHT HAVE BEEN RUNNING WHEN THE
254   '  ERROR OCCURRED.
255
256       CLUSTS% = 0
257       CALLS ABSOLUTE(CLUSTS%, DMA.STOP)
258
259   '  PRINT THE ERROR MESSAGE PASSED IN AS A PARAMETER AND THEN PRINT THE
260   '  ERROR CODE NUMBER
261
262       PRINT ErrorString; ERROR.CODE%
263       END
```

```
264
265     END IF
266
267   END SUB
268
```

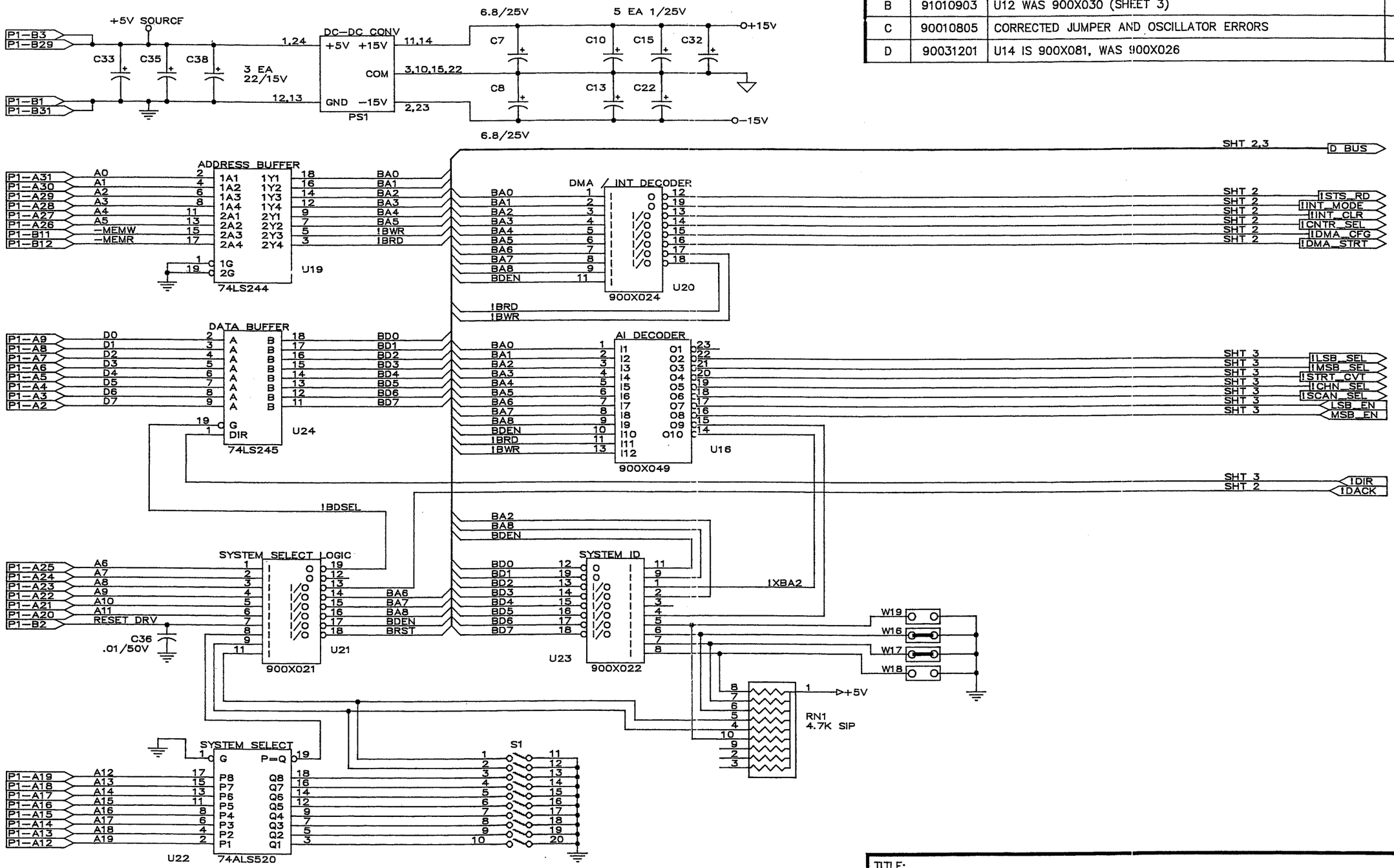
GRAPH.BAS

```
  1   DECLARE SUB WaitKey ()
  2   DECLARE SUB InitWindow (FRAME())
  3   DECLARE SUB PlotWindow (FRAME(), WINDNO)
  4   DECLARE SUB PlotData (FRAME(), Y(), PLOTNO, WINDNO)
  5
  6   CLS : SCREEN 2: KEY OFF
  7
  8   ' FRAME(i,w) is a two-dimensional array containing the coordinates of the
  9   ' corners of each window defined.  Mnemonic indices are defined for the
 10   ' four corners of each window:
 11   '       TOPX and TOPY define the upper left hand corner;
 12   '       BOTX and BOTY the lower right hand corner of the plot windows.
 13   ' WINDS sets the number of windows defined to 9.
 14
 15   ' Y(m,n) is a two dimensional array containing the data sets for each plot
 16   ' where m is the plot number, and n is the data point index.
 17   ' PASSES sets the number of datapoints per plot to 100.
 18
 19   CONST MAXPLOTS = 16: CONST PASSES = 100: CONST MAXWINDS = 9
 20   CONST BOTX = 1: CONST TOPX = 2: CONST BOTY = 3: CONST TOPY = 4
 21
 22   ' The arrays are dimensioned.  This is done once at the beginning of the
 23   ' program.
 24
 25   DIM FRAME(4, MAXWINDS)
 26   DIM Y(MAXPLOTS, PASSES)
 27
 28   ' The following program sets up 8 data sets to demonstrate the plotting
 29   ' routines.
 30
 31   ' First, define the windows to plot data in.
 32
 33   InitWindow FRAME()
 34
 35   ' Set up test data.  Define 8 data sets to plot.
 36
 37   FOR I = 1 TO 100
 38       Y(1, I) = INT(50 * SIN(I * 3.14 / 50))
 39       Y(2, I) = INT(.1 * I * I)
 40       Y(3, I) = Y(1, I) + .1 * Y(2, I)
 41       Y(4, I) = .1 * Y(1, I) * Y(2, I)
 42       Y(5, I) = 20000 * RND(I) - 10000
 43       Y(6, I) = 10 * LOG(I + .5)
 44       Y(7, I) = 25 * COS(I * 3.14 / 25) / (I + 25)
 45       Y(8, I) = INT(I / 20)
 46   NEXT I
 47
 48   ' Finally, plot the data sets.
 49
 50   CLS
 51   LOCATE 10, 32
 52   PRINT "Ready to plot."
 53   LOCATE 12, 14
 54   PRINT "Press any key when you are ready for the next screen."
 55   WaitKey
 56
 57   PLOTNO = 1
 58
 59   FOR SETNO = 1 TO 2
 60       FOR WINDNO = 1 TO 4
 61           PlotData FRAME(), Y(), PLOTNO, WINDNO
 62           PLOTNO = PLOTNO + 1
 63       NEXT WINDNO
 64       WaitKey
 65   NEXT SETNO
 66
 67   WINDNO = 5
 68   FOR PLOTNO = 1 TO 2
 69       PlotData FRAME(), Y(), PLOTNO, WINDNO
 70       WaitKey
 71   NEXT PLOTNO
 72
 73   PLOTNO = 3
 74   FOR SETNO = 1 TO 2
 75       FOR WINDNO = 6 TO 7
```

```
76              PlotData FRAME(), Y(), PLOTNO, WINDNO
77              PLOTNO = PLOTNO + 1
78          NEXT WINDNO
79          WaitKey
80      NEXT SETNO
81
82      FOR WINDNO = 8 TO 9
83          PlotData FRAME(), Y(), PLOTNO, WINDNO
84          PLOTNO = PLOTNO + 1
85      NEXT WINDNO
86
87      WaitKey
88      END
89
90      SUB InitWindow (FRAME())
91
92      ' Define windows to plot in.
93
94      ' Windows 1, 2, 3, and 4 are one-quarter screen windows.
95
96      FRAME(TOPX, 1) = 296: FRAME(TOPY, 1) = 4
97      FRAME(BOTX, 1) = 56: FRAME(BOTY, 1) = 84
98
99      FRAME(TOPX, 2) = 600: FRAME(TOPY, 2) = 4
100     FRAME(BOTX, 2) = 360: FRAME(BOTY, 2) = 84
101
102     FRAME(TOPX, 3) = 296: FRAME(TOPY, 3) = 108
103     FRAME(BOTX, 3) = 56: FRAME(BOTY, 3) = 188
104
105     FRAME(TOPX, 4) = 600: FRAME(TOPY, 4) = 108
106     FRAME(BOTX, 4) = 360: FRAME(BOTY, 4) = 188
107
108     ' Window 5 is a full-screen window.
109
110     FRAME(TOPX, 5) = 616: FRAME(TOPY, 5) = 4
111     FRAME(BOTX, 5) = 56: FRAME(BOTY, 5) = 188
112
113     ' Windows 6 and 7 are half-screen horizontal windows.
114
115     FRAME(TOPX, 6) = 600: FRAME(TOPY, 6) = 4
116     FRAME(BOTX, 6) = 56: FRAME(BOTY, 6) = 84
117
118     FRAME(TOPX, 7) = 600: FRAME(TOPY, 7) = 108
119     FRAME(BOTX, 7) = 56: FRAME(BOTY, 7) = 188
120
121     ' Windows 8 and 9 are half-screen vertical windows.
122
123     FRAME(TOPX, 8) = 296: FRAME(TOPY, 8) = 4
124     FRAME(BOTX, 8) = 56: FRAME(BOTY, 8) = 188
125
126     FRAME(TOPX, 9) = 600: FRAME(TOPY, 9) = 4
127     FRAME(BOTX, 9) = 360: FRAME(BOTY, 9) = 188
128
129     END SUB
130
131     SUB PlotData (FRAME(), Y(), PLOTNO, WINDNO)
132
133     ' Subroutine to scale and plot the data in the previously defined
134     ' window.
135
136     ' First, find minimum and maximum data values, and calculate scaling
137     ' constants to plot the data in a window.
138
139     YMIN = Y(PLOTNO, 1): YMAX = Y(PLOTNO, 1)
140     POINTS = UBOUND(Y, 2)
141
142     FOR I = 2 TO POINTS
143         IF Y(PLOTNO, I) > YMAX THEN
144             YMAX = Y(PLOTNO, I)
145         END IF
146         IF Y(PLOTNO, I) < YMIN THEN
147             YMIN = Y(PLOTNO, I)
148         END IF
149     NEXT I
150
151     X1 = FRAME(BOTX, WINDNO): Y1 = FRAME(BOTY, WINDNO)
152     X2 = FRAME(TOPX, WINDNO): Y2 = FRAME(TOPY, WINDNO)
153
154     ' The following line computes the factors to autoscale the data
155     ' in the Y axis.
156
157     YSPAN = YMAX - YMIN: FYFACT = (Y2 - Y1) / YSPAN
158     FXINC = (X2 - X1) / POINTS
159
160     ' Plot the window, label x and y axes
161
162     PlotWindow FRAME(), WINDNO
```

```
163
164   LOCATE INT(Y2 / 8 + 1), INT(X1 / 8 - 4): PRINT USING "####"; YMAX;
165   LOCATE INT(Y1 / 8 + 1), INT(X1 / 8 - 5): PRINT USING "#####"; YMIN;
166   LOCATE INT(Y1 / 8 + 2), INT(X2 / 8 - 1): PRINT USING "###"; POINTS;
167   LOCATE INT(Y1 / 8 + 2), INT(X1 / 8 + 1): PRINT "0";
168
169   FOLDX = X1
170
171   ' Scale and plot the data
172
173   FOR J = 1 TO POINTS
174
175   ' First, autoscale a new Y point to the plot window
176      NEWY = (Y(PLOTNO, J) - YMIN) * FYFACT + Y1
177
178   ' Then, autoscale a new X point to the plot window
179      FXNEW = FOLDX + FXINC
180
181   ' Next, draw a line from the previous point to the new point
182      IF J > 1 THEN
183         LINE (FOLDX, OLDY)-(FXNEW, NEWY)
184      END IF
185
186   ' Finally, the new XY point becomes the old XY point
187      FOLDX = FXNEW: OLDY = NEWY
188   NEXT J
189   END SUB
190
191   SUB PlotWindow (FRAME(), WINDNO)
192
193   ' Subroutine to draw the plotting window and add tic marks to the axes.
194
195   X1 = FRAME(BOTX, WINDNO): Y1 = FRAME(BOTY, WINDNO)
196   X2 = FRAME(TOPX, WINDNO): Y2 = FRAME(TOPY, WINDNO)
197
198   ' Set up size of tic marks, and clear border in plot window
199   YTIC = 5: XTIC = 10: BORD = 2
200
201   ' Draw the plot window
202   LINE (X1, Y1 + BORD)-(X1, Y2 - BORD)
203   LINE (X1, Y1 + BORD)-(X2, Y1 + BORD)
204   LINE (X2, Y2 - BORD)-(X2, Y1 + BORD)
205   LINE (X2, Y2 - BORD)-(X1, Y2 - BORD)
206
207   ' Add tic marks to the Y axis
208   FYINT = (Y2 - Y1) / 10
209
210   FOR I = 0 TO 10
211      LINE (X1, Y1 + I * FYINT)-(X1 - XTIC, Y1 + I * FYINT)
212   NEXT I
213
214   ' Add tic marks to the X axis
215   FXINT = (X2 - X1) / 10
216
217   FOR I = 0 TO 10
218      LINE (X1 + I * FXINT, Y1 + BORD)-(X1 + I * FXINT, Y1 + YTIC + BORD)
219   NEXT I
220   END SUB
221
222   SUB WaitKey
223   DO
224      K$ = INKEY$
225   LOOP WHILE K$ = ""
226   CLS
227   END SUB
228
```

LTR	ECO	REVISIONS	DATE	APRV'D
A	89091902	RE-RELEASED UNDER NEW P/N (OLD P/N WAS 923S002)		
B	91010903	U12 WAS 900X030 (SHEET 3)		
C	90010805	CORRECTED JUMPER AND OSCILLATOR ERRORS		
D	90031201	U14 IS 900X081, WAS 900X026		



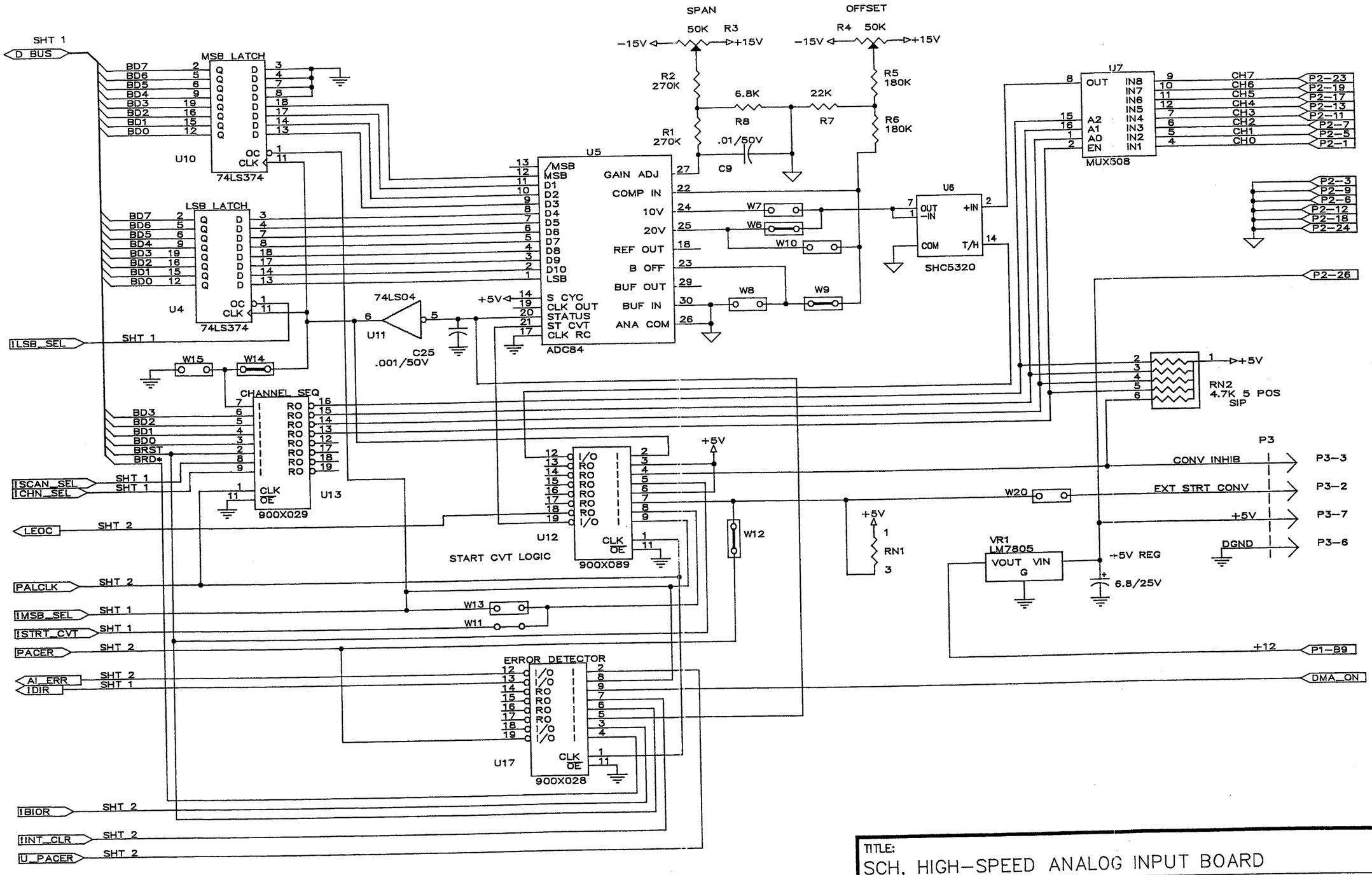
TITLE: SCH, HIGH-SPEED ANALOG INPUT BOARD			PLOT DATE: 17.JUN.91	
PRODUCT NO: 91W-1	VOL LABEL: 923S008.1-1	DOCUMENT NO: 923S008 SCH	REV: D	SHEET: 1 / 4
	FILE NAME: 923S008A.SCH			

LTR	ECO	REVISIONS	DATE	APRV'D

PACER SHT 3
LEOC SHT 3
!STS_RD SHT 1
D_BUS SHT 1
STATUS BUFFER
U8
74LS244
BD7 BD6 BD5 BD4 BD3 BD2 BD1 BD0
2Y4 2Y3 2Y2 2Y1 1Y4 1Y3 1Y2 1Y1
2A4 2A3 2A2 2A1 1A4 1A3 1A2 1A1
2G 1G
W4 W5
IRQ LATCH
U18
900X025
PCR EI TC
I/O RO RO RO RO RO RO I/O
CLK OE
RATE GENERATOR BUFFER
U11 74LS04
RG OUT P3-1
EXT INT P3-5
+5V
RN1-2
W32 IRQ2 P1-4B
W22 IRQ3 P1-25B
W23 IRQ4 P1-24B
W24 IRQ5 P1-23B
W25 IRQ6 P1-22B
W26 IRQ7 P1-21B
/EXT INT
U2 74LS74
TRIGGER FF
!INT_MODE SHT 1
!INT_CLR SHT 1, 3
BRST
W3 W2 W1 N.C.
U3 8 MHz OSCILLATOR
SHT 3 AI_ERR
DMA_ON
PALCLK
74LS04
CLK P1-20B
SHT 3 PALCLK
PACER SHT 3
COUNTER
8254
U9
BD0 BD1 BD2 BD3 BD4 BD5 BD6 BD7
D0 D1 D2 D3 D4 D5 D6 D7
CLK0 G0 OUT0 CLK1 G1 OUT1 CLK2 G2 OUT2
BRD* BWR* BA0 BA1
RD WR A0 A1 CS
!CNTR_SEL SHT 1
U_PACER
DMA CONFIG
U1
74LS374
BD4 BD3 BD5 BD0
DELAY EN
PACER EN
DMA EN
DMA MODE
!DMA_CFG SHT 1
BRST
LSB_EN SHT 1
MSB_EN SHT 1
!DMA_STRT SHT 1
DMA CONTROLLER SHT 4
DMA_ON LEOC MYTC DELAY !DACK DMA_EN DMA_MODE BRST LSB_EN MSB_EN !DMA_START
PALCLK DREQ TC !IOR !BIOR
R9 +5V
SHT 1 !DACK
W28 -DACK1 P1-17B
W21 -DACK2 P1-26B
W30 -DACK3 P1-15B
C31 22PF
W27 DRQ1 P1-18B
W31 DRQ2 P1-6B
W29 DRQ3 P1-16B
TC P1-27B
C34 470PF
-IOR P1-14B
SHT 3 !BIOR

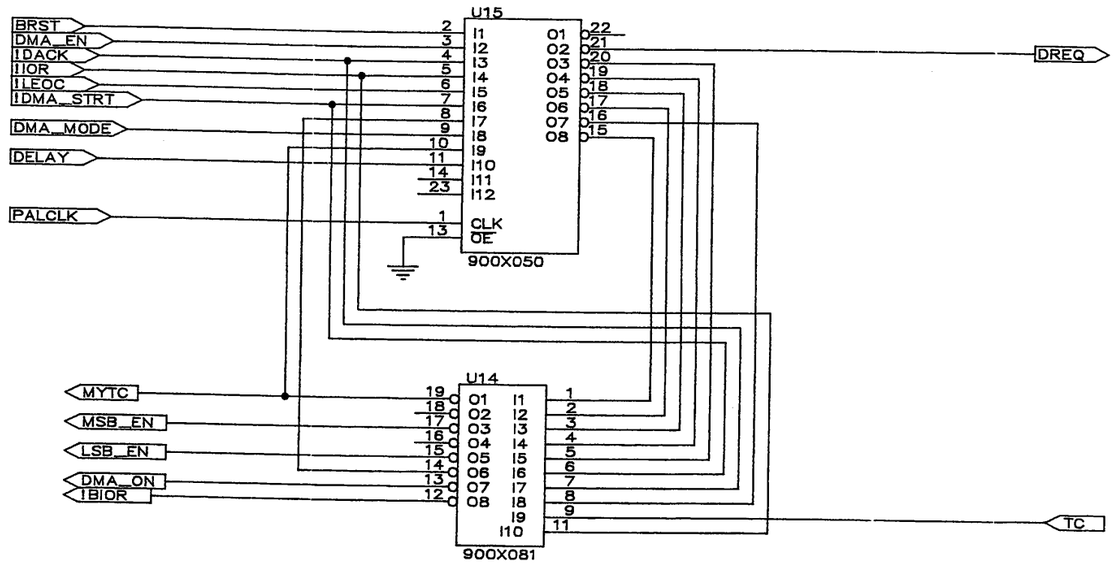
TITLE: SCH, HIGH-SPEED ANALOG INPUT BOARD			PLOT DATE: 17.JUN.91	
PRODUCT NO: 91W-1	VOL LABEL: 923S008.1-1	DOCUMENT NO: 923S008 SCH	REV: D	SHEET: 2 / 4
	FILE NAME: 923S008B.SCH			

LTR	ECO	REVISIONS	DATE	APRV'D



TITLE: SCH, HIGH−SPEED ANALOG INPUT BOARD			PLOT DATE: 17.JUN.91
PRODUCT NO: 91W−1	VOL LABEL: 923S008.1−1	DOCUMENT NO: 923S008 SCH	REV: D / SHEET: 3 / 4
	FILE NAME: 923S008C.SCH		

LTR	ECO	REVISIONS	DATE	APRV'D



IC POWER AND GROUND TABLE

	+5V	DGND	+15V	−15V	AGND	SHT
U1	20	10				2
U2	14	7				2
U3	14	7				2
U4	20	10				3
U5	16	15	28	31	26	3
U6			9	5	6,13	3
U7			13	3	14	3
U8	20	10				2
U9	24	12				2
U10	20	10				3
U11	14	7				2
U12	20	10				2,3
U13	20	10				3
U14	20	10				3
U15	24	12				4
U16	24	12				4
U17	20	10				3
U18	20	10				2
U19	20	10				1
U20	20	10				1
U21	20	10				1
U22	20	10				1
U23	20	10				1
U24	20	10				1

TITLE: SCH, HIGH−SPEED ANALOG INPUT BOARD				PLOT DATE: 17.JUN.91
PRODUCT NO: 91W−1	VOL LABEL: 923S008.1−1	DOCUMENT NO: 923S008 SCH	REV: D	SHEET: 4 / 4
	FILE NAME: 923S008D.SCH			